

CODEC Messages

CODEC procedure messages can be sent to a CODEC procedure as a result of an MMIO function call. For example, the `MMIOM_CODEC_OPEN` message is sent to a CODEC procedure when `mmioOpen` is called.

CODEC procedures can also be loaded and called directly, without using `mmioSendMessage`. The `mmioLoadCODECProc` function loads a CODEC procedure and returns the entry point. A call to this entry point with an `MMIOM_CODEC_OPEN` message returns an `HCODEC`, which is used to identify the open instance on all other calls to the CODEC.

The following syntax is required for a direct CODEC procedure call:

```
typedef LONG (APIENTRY CODECPROC) (PHCODEC phcodec,
                                   USHORT usMsg,
                                   LONG lParam1,
                                   LONG lParam2);

typedef CODECPROC *PCODECPROC;
```

Note that OS/2 multimedia currently supports image and video CODECs. OS/2 multimedia does provide a set of audio CODECs, but does not provide a public interface to install new audio CODECs or to interface to audio CODECs directly.

All CODEC procedures must support the following messages. However, if a CODEC is a compressor only, `MMIOM_CODEC_DECOMPRESS` does not need to be supported. The same is true if a CODEC is a decompressor only; `MMIOM_CODEC_COMPRESS` does not need to be supported.

Message	Description
<code>MMIOM_CODEC_CLOSE</code>	Requests close of instance specified by <i>phCODEC</i> .
<code>MMIOM_CODEC_COMPRESS</code>	Requests CODEC to compress the data.
<code>MMIOM_CODEC_DECOMPRESS</code>	Requests CODEC to decompress the data.
<code>MMIOM_CODEC_OPEN</code>	Requests CODEC to open an instance.
<code>MMIOM_CODEC_QUERYNAME</code>	Requests the name of the CODEC.
<code>MMIOM_CODEC_QUERYNAMELENGTH</code>	Requests the length of the CODEC name.

MMIOM_CODEC_CLOSE

MMIOM_CODEC_CLOSE Parameter - phCODEC

phCODEC (`PHCODEC`)
Pointer to CODEC handle.

MMIOM_CODEC_CLOSE Parameter - usMsg

usMsg ([USHORT](#))
Set to MMIOM_CODEC_CLOSE.

MMIOM_CODEC_CLOSE Parameter - IParam1

IParam1 ([LONG](#))
This parameter is not used.

MMIOM_CODEC_CLOSE Parameter - IParam2

IParam2 ([ULONG](#))
This parameter is not used.

MMIOM_CODEC_CLOSE Return Value - rc

rc ([ULONG](#))
Return codes indicating success or failure:

MMIO_SUCCESS	The specified instance was closed successfully.
MMIO_ERROR	An error code is returned.

MMIOM_CODEC_CLOSE - Description

This message requests that a CODEC opened by [MMIOM_CODEC_OPEN](#) be closed.

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

usMsg ([USHORT](#))
Set to MMIOM_CODEC_CLOSE.

IParam1 ([LONG](#))
This parameter is not used.

IParam2 ([ULONG](#))
This parameter is not used.

rc ([ULONG](#))
Return codes indicating success or failure:

MMIO_SUCCESS	The specified instance was closed successfully.
MMIO_ERROR	An error code is returned.

MMIOM_CODEC_CLOSE - Topics

Select an item:
[Description](#)
[Returns](#)
[Glossary](#)

MMIOM_CODEC_COMPRESS

MMIOM_CODEC_COMPRESS Parameter - phCODEC

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

MMIOM_CODEC_COMPRESS Parameter - usMsg

usMsg ([USHORT](#))
Set to MMIOM_CODEC_COMPRESS.

MMIOM_CODEC_COMPRESS Parameter - IParam1

IParam1 ([LONG](#))

Pointer to an [MMCOMPRESS](#) structure.

MMIOM_CODEC_COMPRESS Parameter - IParam2

IParam2 ([LONG](#))

This parameter is not used.

MMIOM_CODEC_COMPRESS Return Value - rc

rc ([ULONG](#))

Return codes indicating success or failure:

MMIO_SUCCESS

The data is compressed successfully.

MMIO_ERROR

An error code is returned.

MMIOM_CODEC_COMPRESS - Description

This message requests that data be compressed by the CODEC.

phCODEC ([PHCODEC](#))

Pointer to CODEC handle.

usMsg ([USHORT](#))

Set to MMIOM_CODEC_COMPRESS.

IParam1 ([LONG](#))

Pointer to an [MMCOMPRESS](#) structure.

IParam2 ([LONG](#))

This parameter is not used.

rc ([ULONG](#))

Return codes indicating success or failure:

MMIO_SUCCESS

The data is compressed successfully.

MMIO_ERROR

An error code is returned.

MMIOM_CODEC_COMPRESS - Topics

Select an item:

[Description](#)

[Returns](#)

[Glossary](#)

MMIOM_CODEC_DECOMPRESS

MMIOM_CODEC_DECOMPRESS Parameter - phCODEC

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

MMIOM_CODEC_DECOMPRESS Parameter - usMsg

usMsg ([USHORT](#))
Set to MMIOM_CODEC_DECOMPRESS.

MMIOM_CODEC_DECOMPRESS Parameter - IParam1

IParam1 ([LONG](#))
Pointer to the [MMVIDEODECOMPRESS](#) structure.

MMIOM_CODEC_DECOMPRESS Parameter - IParam2

IParam2 ([LONG](#))
This parameter is not used.

MMIOM_CODEC_DECOMPRESS Return Value - rc

rc ([ULONG](#))
Return codes indicating success or failure:

MMIO_SUCCESS
The data is requested successfully.

MMIO_ERROR
An error code is returned.

MMIOERR_ERROR_IN_FRAME_DATA
An invalid frame was detected.

MMIOERR_INVALID_DIM_ALIGN
An invalid pixel alignment was detected.

MMIOM_CODEC_DECOMPRESS - Description

This message requests the CODEC Proc to decompress the data.

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

usMsg ([USHORT](#))
Set to MMIOM_CODEC_DECOMPRESS.

IParam1 ([LONG](#))
Pointer to the [MMVIDEODECOMPRESS](#) structure.

IParam2 ([LONG](#))
This parameter is not used.

rc ([ULONG](#))
Return codes indicating success or failure:

MMIO_SUCCESS
The data is requested successfully.

MMIO_ERROR
An error code is returned.

MMIOERR_ERROR_IN_FRAME_DATA
An invalid frame was detected.

MMIOERR_INVALID_DIM_ALIGN
An invalid pixel alignment was detected.

MMIOM_CODEC_DECOMPRESS - Topics

Select an item:
[Description](#)
[Returns](#)
[Glossary](#)

MMIOM_CODEC_OPEN

MMIOM_CODEC_OPEN Parameter - phCODEC

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

MMIOM_CODEC_OPEN Parameter - usMsg

usMsg ([USHORT](#))
Set to MMIOM_CODEC_OPEN.

MMIOM_CODEC_OPEN Parameter - IParam1

IParam1 ([LONG](#))
Pointer to an [CODECOPEN](#) structure.

MMIOM_CODEC_OPEN Parameter - IParam2

IParam2 ([LONG](#))
This parameter is not used.

MMIOM_CODEC_OPEN Return Value - rc

rc ([ULONG](#))
Return codes indicating success or failure:

[MMIO_SUCCESS](#)
The specified instance is opened successfully.

MMIO_ERROR

An error code is returned.

MMIOM_CODEC_OPEN - Description

This message requests a CODEC procedure instance be opened. On completion, it returns a CODEC handle in *phCODEC*.

phCODEC ([PHCODEC](#))

Pointer to CODEC handle.

usMsg ([USHORT](#))

Set to MMIOM_CODEC_OPEN.

IParam1 ([LONG](#))

Pointer to an [CODECOPEN](#) structure.

IParam2 ([LONG](#))

This parameter is not used.

rc ([ULONG](#))

Return codes indicating success or failure:

MMIO_SUCCESS

The specified instance is opened successfully.

MMIO_ERROR

An error code is returned.

MMIOM_CODEC_OPEN - Remarks

Typically the *phCODEC* parameter is set to NULL. If multiple CODEC algorithms are all installed in one DLL with separate entry points, subsequent open calls can have the *phCODEC* set to the first returned handle. This allows the CODEC procedure to share the same movie data structure among the different algorithms.

MMIOM_CODEC_OPEN - Topics

Select an item:

[Description](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

MMIOM_CODEC_QUERYNAME

MMIOM_CODEC_QUERYNAME Parameter - phCODEC

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

MMIOM_CODEC_QUERYNAME Parameter - usMsg

usMsg ([USHORT](#))
Set to MMIOM_CODEC_QUERYNAME.

MMIOM_CODEC_QUERYNAME Parameter - IParam1

IParam1 ([LONG](#))
Pointer to the name-string buffer.

MMIOM_CODEC_QUERYNAME Parameter - IParam2

IParam2 ([LONG](#))
Pointer to the name-string buffer length in bytes. On output, the actual number of bytes read is returned.

MMIOM_CODEC_QUERYNAME Return Value - rc

rc ([ULONG](#))
Return codes indicating success or failure:

MMIO_SUCCESS	The information is requested successfully.
MMIO_ERROR	An error code is returned.

MMIOM_CODEC_QUERYNAME - Description

This message is sent to the CODEC procedure by [mmioQueryCODECName](#) to request the CODEC name.

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

usMsg ([USHORT](#))
Set to MMIOM_CODEC_QUERYNAME.

IParam1 ([LONG](#))
Pointer to the name-string buffer.

IParam2 ([LONG](#))
Pointer to the name-string buffer length in bytes. On output, the actual number of bytes read is returned.

rc ([ULONG](#))
Return codes indicating success or failure:

MMIO_SUCCESS	The information is requested successfully.
MMIO_ERROR	An error code is returned.

MMIOM_CODEC_QUERYNAME - Topics

Select an item:
[Description](#)
[Returns](#)
[Glossary](#)

MMIOM_CODEC_QUERYNAMELENGTH

MMIOM_CODEC_QUERYNAMELENGTH Parameter - phCODEC

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

MMIOM_CODEC_QUERYNAMELENGTH Parameter - usMsg

usMsg ([USHORT](#))
Set to MMIOM_CODEC_QUERYNAMELENGTH.

MMIOM_CODEC_QUERYNAMELENGTH Parameter - IParam1

IParam1 ([LONG](#))
Pointer to the length in bytes of the CODEC name string.

MMIOM_CODEC_QUERYNAMELENGTH Parameter - IParam2

IParam2 ([LONG](#))
This parameter is not used.

MMIOM_CODEC_QUERYNAMELENGTH Return Value - rc

rc ([ULONG](#))
Return codes indicating success or failure:

MMIO_SUCCESS	The information is requested successfully.
MMIO_ERROR	An error code is returned.

MMIOM_CODEC_QUERYNAMELENGTH - Description

This message is sent to the CODEC procedure by [mmioQueryCODECNameLength](#) to request the length of the CODEC name.

phCODEC ([PHCODEC](#))
Pointer to CODEC handle.

usMsg ([USHORT](#))
Set to MMIOM_CODEC_QUERYNAMELENGTH.

IParam1 ([LONG](#))
Pointer to the length in bytes of the CODEC name string.

IParam2 ([LONG](#))
This parameter is not used.

rc ([ULONG](#))
Return codes indicating success or failure:

[MMIO_SUCCESS](#)
The information is requested successfully.

[MMIO_ERROR](#)
An error code is returned.

MMIOM_CODEC_QUERYNAMELENGTH - Topics

Select an item:

- [Description](#)
- [Returns](#)
- [Glossary](#)

DIVE Functions

Direct interface video extensions (DIVE) provide optimized blitting performance for motion video subsystems and applications that perform rapid screen updates in the OS/2 PM and full-screen environments. Using the DIVE interface, applications can either write directly to video memory or use the DIVE blitter. The DIVE blitter will take advantage of acceleration hardware when present and applicable to the function being performed.

The following table describes the DIVE functions.

Function	Description
DiveAcquireFrameBuffer	Allows the frame buffer to be serialized.
DiveAllocImageBuffer	Allocates a buffer to contain an image.
DiveBeginImageBufferAccess	Begins access to the image buffer.
DiveBlitImage	Transfers an image from the source to its destination.
DiveBlitImageLines	Transfers only the changed lines of an image from the source to its destination.
DiveCalcFrameBufferAddress	Calculates linear frame buffer address.
DiveClose	Closes instance.
DiveDeacquireFrameBuffer	Releases exclusive access to the frame buffer.
DiveEndImageBufferAccess	Ends access to the image buffer.
DiveFreeImageBuffer	Deallocates an image buffer allocated by DiveAllocImageBuffer .

<code>DiveOpen</code>	Opens a display engine instance.
<code>DiveQueryCaps</code>	Queries display capabilities.
<code>DiveSetDestinationPalette</code>	Sets palette associated with the destination of <code>DiveBlitImage</code> .
<code>DiveSetupBlitter</code>	Sets up blitter operations.
<code>DiveSetSourcePalette</code>	Sets the palette associated with source data.
<code>DiveSwitchBank</code>	Selects VRAM bank for bank-switched displays.

DiveAcquireFrameBuffer

DiveAcquireFrameBuffer - Syntax

This function allows the frame buffer to be serialized. The frame buffer is locked for this instance. No other instance can acquire or switch aperture banks until this instance has deacquired the frame buffer.

```
#include <dive.h>

HDIVE    hDiveInst; /* Display engine DIVE instance. */
PRECTL   prectlDst; /* Destination rect of the output */
ULONG    rc;        /* Return codes. */

rc = DiveAcquireFrameBuffer(hDiveInst, prectlDst);
```

DiveAcquireFrameBuffer Parameter - hDiveInst

hDiveInst (`HDIVE`) - input
Display engine DIVE instance.

DiveAcquireFrameBuffer Parameter - prectlDst

prectlDst (`PRECTL`) - input
Indicates the destination rectangle of the output area. On bank-switched systems, the bank is automatically switched to the topmost bank in the specified screen destination rectangle.

DiveAcquireFrameBuffer Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_ACQUIRE_FAILED

The acquire action did not complete successfully.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_NO_DIRECT_ACCESS

The display adapter, display driver, or current video mode does not support direct-to-screen access.

DiveAcquireFrameBuffer - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

prectlDst ([PRECTL](#)) - input

Indicates the destination rectangle of the output area. On bank-switched systems, the bank is automatically switched to the topmost bank in the specified screen destination rectangle.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_ACQUIRE_FAILED

The acquire action did not complete successfully.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_NO_DIRECT_ACCESS

The display adapter, display driver, or current video mode does not support direct-to-screen access.

DiveAcquireFrameBuffer - Remarks

If another instance has the frame buffer acquired, the call will block until the frame buffer can be acquired.

The DiveAcquireFrameBuffer call and the corresponding [DiveDeacquireFrameBuffer](#) call are not necessary if you are using [DiveBlitImage](#) rather than writing directly to the screen with the *ppFrameBuffer* from [DiveOpen](#).

DiveAcquireFrameBuffer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DiveAllocImageBuffer

DiveAllocImageBuffer - Syntax

This function allocates a buffer to contain an image. Source data buffers passed to [DiveBlitImage](#) must be allocated with this function in order to take advantage of hardware acceleration features of some display hardware.

```
#include <dive.h>

HDIVE      hDiveInst;          /* Display engine DIVE instance. */
PULONG     pulBufferNumber;    /* Buffer number allocated. */
FOURCC     fccColorSpace;     /* Pel format of image color space. */
ULONG      ulWidth;           /* Width of DIVE memory to allocate. */
ULONG      ulHeight;          /* Height of DIVE memory to allocate. */
ULONG      ulLineSizeBytes;    /* Suggested scan line size. */
PBYTE      pbImageBuffer;     /* Image buffer. */
ULONG      rc;                /* Return codes. */

rc = DiveAllocImageBuffer(hDiveInst, pulBufferNumber,
                          fccColorSpace, ulWidth, ulHeight, ulLineSizeBytes,
                          pbImageBuffer);
```

DiveAllocImageBuffer Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveAllocImageBuffer Parameter - pulBufferNumber

pulBufferNumber ([PULONG](#)) - output
Buffer number allocated for this instance.

DiveAllocImageBuffer Parameter - fccColorSpace

fccColorSpace ([FOURCC](#)) - input
Indicates pel format of image color space this buffer will contain.

DiveAllocImageBuffer Parameter - ulWidth

ulWidth ([ULONG](#)) - input
Indicates width (in pels) of DIVE memory to allocate.

DiveAllocImageBuffer Parameter - ulHeight

ulHeight ([ULONG](#)) - input
Indicates height (in pels) of DIVE memory to allocate.

DiveAllocImageBuffer Parameter - ulLineSizeBytes

ulLineSizeBytes ([ULONG](#)) - input
Suggested scan line size for allocated buffer. Setting *ulLineSizeBytes* to 0 allows DIVE to calculate the optimum line size.

DiveAllocImageBuffer Parameter - pblImageBuffer

pblImageBuffer ([PBYTE](#)) - input
Use *pblImageBuffer* to associate a buffer (that has already been allocated by some means other than DIVE) to a DIVE buffer number. You must still call [DiveFreeImageBuffer](#) to free up the associated buffer and deallocate the buffer using the same means in which it was allocated.

DiveAllocImageBuffer Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS
If the function succeeds, 0 is returned.

DIVE_ERR_ALLOCATION_ERROR

The hardware-blitter memory allocation failed.

DIVE_ERR_INVALID_BUFFER_NUMBER

The parameter *pulBufferNumber* must either point to zero (a newly associated buffer) or a currently associated buffer number (to reassociate the buffer to a new pointer).

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_INVALID_LINESIZE

The scan line size specified in the *ulLineSizeBytes* parameter is invalid.

DIVE_ERR_SOURCE_FORMAT

The source format is not a recognized FOURCC.

DIVE_ERR_TOO_MANY_INSTANCES

There are not enough resources for another DIVE instance.

DiveAllocImageBuffer - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

pulBufferNumber ([PULONG](#)) - output

Buffer number allocated for this instance.

fccColorSpace ([FOURCC](#)) - input

Indicates pel format of image color space this buffer will contain.

ulWidth ([ULONG](#)) - input

Indicates width (in pels) of DIVE memory to allocate.

ulHeight ([ULONG](#)) - input

Indicates height (in pels) of DIVE memory to allocate.

ulLineSizeBytes ([ULONG](#)) - input

Suggested scan line size for allocated buffer. Setting *ulLineSizeBytes* to 0 allows DIVE to calculate the optimum line size.

pblImageBuffer ([PBYTE](#)) - input

Use *pblImageBuffer* to associate a buffer (that has already been allocated by some means other than DIVE) to a DIVE buffer number. You must still call [DiveFreeImageBuffer](#) to free up the associated buffer and deallocate the buffer using the same means in which it was allocated.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_ALLOCATION_ERROR

The hardware-blitter memory allocation failed.

DIVE_ERR_INVALID_BUFFER_NUMBER

The parameter *pulBufferNumber* must either point to zero (a newly associated buffer) or a currently associated buffer number (to reassociate the buffer to a new pointer).

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_INVALID_LINESIZE

The scan line size specified in the *ulLineSizeBytes* parameter is invalid.

DIVE_ERR_SOURCE_FORMAT

The source format is not a recognized FOURCC.

DIVE_ERR_TOO_MANY_INSTANCES

There are not enough resources for another DIVE instance.

DiveAllocImageBuffer - Remarks

This function will allocate a buffer to contain an image. The buffer may be allocated in either system memory or in VRAM, depending on a number of factors. The entire image buffer will be allocated in video memory if all of the following conditions are met:

- Display hardware that accelerates certain operations is installed, and that hardware requires the source image data to reside in video memory.
- The source image data is in a format that the hardware accepts as an input for accelerating certain operations.
- Sufficient VRAM is available to hold the image.

If any of these conditions are not met, the buffer will be allocated in system memory.

If *pbImageBuffer* is 0, DIVE will allocate a buffer for the specified size and FOURCC. If *pbImageBuffer* is a pointer to a non-DIVE-allocated buffer, DIVE will associate a buffer number to that pointer.

Even though no memory is allocated by DiveAllocImageBuffer when user-allocated buffers are associated, [DiveFreeImageBuffer](#) should be called to release the buffer association to avoid using up available buffer indexes. The specified line size will be used if a buffer is allocated in system memory, or if a user buffer is associated. If the specified line size is zero, the allocated line size is rounded up to the nearest ULONG boundary.

Because only one blitter setup is supported at any time for an instance, if multiple source buffers are allocated in an instance, they must be of the same size and color space to work with [DiveBlitImage](#).

Note: If hardware acceleration is present and the *fccColorSpace* format is supported as input to the hardware, the allocation will be performed with a `devescape_imgbufalloc` call to the display driver. If the format is not supported by hardware or if hardware acceleration is not available, the buffer will be allocated in system memory using `DosAllocMem`.

DiveAllocImageBuffer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DiveBeginImageBufferAccess

DiveBeginImageBufferAccess - Syntax

This function must be called before reading or writing data in a buffer allocated with [DiveAllocImageBuffer](#). Depending on the underlying

hardware implementation, this function might result in the frame buffer access being serialized (see [DiveAcquireFrameBuffer](#)). Once the caller has completed accessing (reading or writing) the image buffer, [DiveEndImageBufferAccess](#) must be called. Note that the pointer to the allocated buffer and line size are returned by this function, and these values might change from one image buffer access to the next.

```
#include <dive.h>

HDIVE      hDiveInst;           /* Display engine DIVE instance. */
ULONG      ulBufferNumber;      /* Buffer number being requested. */
PBYTE      *ppbImageBuffer;     /* Pointer to pointer. */
PULONG      pulBufferScanLineBytes; /* Pointer to scan line size. */
PULONG      pulBufferScanLines;  /* Pointer to # of scan lines. */
ULONG      rc;                  /* Return codes. */

rc = DiveBeginImageBufferAccess(hDiveInst,
                                ulBufferNumber, ppbImageBuffer, pulBufferScanLineBytes,
                                pulBufferScanLines);
```

DiveBeginImageBufferAccess Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveBeginImageBufferAccess Parameter - ulBufferNumber

ulBufferNumber ([ULONG](#)) - input
DIVE instance buffer number being requested.

DiveBeginImageBufferAccess Parameter - ppbImageBuffer

ppbImageBuffer ([PBYTE *](#)) - output
Pointer to pointer to allocated image buffer on return.

DiveBeginImageBufferAccess Parameter - pulBufferScanLineBytes

pulBufferScanLineBytes ([PULONG](#)) - output
Pointer to scan line size of allocated memory. On some hardware, this memory might not be allocated in a contiguous fashion.

DiveBeginImageBufferAccess Parameter -

pulBufferScanLines

pulBufferScanLines ([PULONG](#)) - output
Pointer to number of scan lines in the image buffer.

DiveBeginImageBufferAccess Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS
If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_BUFFER_NUMBER
The *ulBufferNumber* parameter must be a previously allocated or associated buffer number before it can be accessed.

DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveBeginImageBufferAccess - Parameters

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

ulBufferNumber ([ULONG](#)) - input
DIVE instance buffer number being requested.

ppblImageBuffer ([PBYTE *](#)) - output
Pointer to pointer to allocated image buffer on return.

pulBufferScanLineBytes ([PULONG](#)) - output
Pointer to scan line size of allocated memory. On some hardware, this memory might not be allocated in a contiguous fashion.

pulBufferScanLines ([PULONG](#)) - output
Pointer to number of scan lines in the image buffer.

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS
If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_BUFFER_NUMBER
The *ulBufferNumber* parameter must be a previously allocated or associated buffer number before it can be accessed.

DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveBeginImageBufferAccess - Remarks

This function has no effect if the image buffer was allocated in system memory.

DiveBeginImageBufferAccess - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DiveBlitImage

DiveBlitImage - Syntax

This function is called to transfer an image from a source to a destination, using parameters specified by [DiveSetupBlitter](#). The internal operation of this function varies greatly, depending on whether hardware is present; the screen is bank-switched and the image is scaled or clipped.

```
#include <dive.h>

HDIVE    hDiveInst;        /* Display engine DIVE instance. */
ULONG    ulSrcBufNumber;   /* Buffer containing source data. */
ULONG    ulDstBufNumber;   /* Identifies buffer number. */
ULONG    rc;               /* Return codes. */

rc = DiveBlitImage(hDiveInst, ulSrcBufNumber,
                  ulDstBufNumber);
```

DiveBlitImage Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveBlitImage Parameter - ulSrcBufNumber

ulSrcBufNumber ([ULONG](#)) - input

Indicates DIVE instance buffer number of the buffer containing the source data. This value is returned by [DiveAllocImageBuffer](#).

DiveBlitImage Parameter - ulDstBufNumber

ulDstBufNumber ([ULONG](#)) - input

The following buffer numbers are reserved for blitting to the screen.

DIVE_BUFFER_SCREEN

Results in the image being transferred to either the graphics plane buffer or the alternate plane buffer, based on whether an alternate buffer exists and the suitability of the overlay plane to accelerate the scaling of the image. If DIVE chooses to use the alternate buffer, it will also paint the overlay "key" color on the graphics plane. This automatic painting does not occur if the alternate plane is explicitly specified.

DIVE_BUFFER_GRAPHICS_PLANE

Results in the image being transferred to the graphics plane.

DIVE_BUFFER_ALTERNATE_PLANE

Results in the image being transferred to the alternate (for example, overlay) plane. If your hardware does not support such a plane, this is an error.

DiveBlitImage Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_ACQUIRE_FAILED

The acquire action did not complete successfully.

DIVE_ERR_BLITTER_NOT_SETUP

[DiveSetupBlitter](#) must be called before a call is made to DiveBlitImage.

DIVE_ERR_BUFFER_NOT_ACCESSED

The buffer has not been accessed (occurs on systems with accelerator-enabled hardware).

DIVE_ERR_FATAL_EXCEPTION

DIVE is unable to register the exception handler to handle bank switching.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveBlitImage - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

ulSrcBufNumber ([ULONG](#)) - input

Indicates DIVE instance buffer number of the buffer containing the source data. This value is returned by [DiveAllocImageBuffer](#).

ulDstBufNumber ([ULONG](#)) - input

The following buffer numbers are reserved for blitting to the screen.

DIVE_BUFFER_SCREEN

Results in the image being transferred to either the graphics plane buffer or the alternate plane buffer, based on whether an alternate buffer exists and the suitability of the overlay plane to accelerate the scaling of the image. If DIVE chooses to use the alternate buffer, it will also paint the overlay "key" color on the graphics plane. This automatic painting does not occur if the alternate plane is explicitly specified.

DIVE_BUFFER_GRAPHICS_PLANE

Results in the image being transferred to the graphics plane.

DIVE_BUFFER_ALTERNATE_PLANE

Results in the image being transferred to the alternate (for example, overlay) plane. If your hardware does not support such a plane, this is an error.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_ACQUIRE_FAILED

The acquire action did not complete successfully.

DIVE_ERR_BLITTER_NOT_SETUP

[DiveSetupBlitter](#) must be called before a call is made to [DiveBlitImage](#).

DIVE_ERR_BUFFER_NOT_ACCESSED

The buffer has not been accessed (occurs on systems with accelerator-enabled hardware).

DIVE_ERR_FATAL_EXCEPTION

DIVE is unable to register the exception handler to handle bank switching.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveBlitImage - Remarks

The buffer passed as the source on [DiveBlitImage](#) should be allocated with [DiveAllocImageBuffer](#). If the source buffer data format is a hardware-supported input format, [DiveAllocImageBuffer](#) attempts to allocate the buffer in video memory on devices that require the input image to the hardware to reside in video memory.

The buffer passed as the destination on [DiveBlitImage](#) should be allocated with [DiveAllocImageBuffer](#) or be one of the predefined buffer numbers for one of the display planes (for example, DIVE_BUFFER_SCREEN).

Note: The screen cannot be used as the source for blitting operations.

DiveBlitImage - Related Functions

- [DiveBlitImageLines](#)
- [DiveSetupBlitter](#)

DiveBlitImage - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

DiveBlitImageLines

DiveBlitImageLines - Syntax

This function is called to blit only the changed lines in the source image by using [DiveBlitImage](#). The changed lines are specified with the *pbLineMask* parameter.

```
#include <dive.h>

HDIVE    hDiveInst;    /* Display engine DIVE instance. */
ULONG    ulSrcBufNumber; /* Buffer containing source data. */
ULONG    ulDstBufNumber; /* Identifies buffer number. */
PBYTE    pbLineMask;   /* Line mask pointer. */
ULONG    rc;           /* Return codes. */

rc = DiveBlitImageLines(hDiveInst, ulSrcBufNumber,
                        ulDstBufNumber, pbLineMask);
```

DiveBlitImageLines Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveBlitImageLines Parameter - ulSrcBufNumber

ulSrcBufNumber ([ULONG](#)) - input
Indicates DIVE instance buffer number of the buffer containing the source data. This value is returned by [DiveAllocImageBuffer](#).

DiveBlitImageLines Parameter - ulDstBufNumber

ulDstBufNumber ([ULONG](#)) - input

The following buffer numbers are reserved for blitting to the screen.

DIVE_BUFFER_SCREEN

Results in the image being transferred to either the graphics plane buffer or the alternate plane buffer, based on whether an alternate buffer exists and the suitability of the overlay plane to accelerate the scaling of the image. If DIVE chooses to use the alternate buffer, it will also paint the overlay "key" color on the graphics plane. This automatic painting does not occur if the alternate plane is explicitly specified.

DIVE_BUFFER_GRAPHICS_PLANE

Results in the image being transferred to the graphics plane.

DIVE_BUFFER_ALTERNATE_PLANE

Results in the image being transferred to the alternate (for example, overlay) plane. If your hardware does not support such a plane, this is an error.

DiveBlitImageLines Parameter - pbLineMask

pbLineMask ([PBYTE](#)) - input

Indicates the line mask pointer for the line changed.

DiveBlitImageLines Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_BLITTER_NOT_SETUP

[DiveSetupBlitter](#) must be called before a call is made to DiveBlitImageLines.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveBlitImageLines - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

ulSrcBufNumber ([ULONG](#)) - input

Indicates DIVE instance buffer number of the buffer containing the source data. This value is returned by [DiveAllocImageBuffer](#).

ulDstBufNumber ([ULONG](#)) - input

The following buffer numbers are reserved for blitting to the screen.

DIVE_BUFFER_SCREEN

Results in the image being transferred to either the graphics plane buffer or the alternate plane buffer, based on

whether an alternate buffer exists and the suitability of the overlay plane to accelerate the scaling of the image. If DIVE chooses to use the alternate buffer, it will also paint the overlay "key" color on the graphics plane. This automatic painting does not occur if the alternate plane is explicitly specified.

DIVE_BUFFER_GRAPHICS_PLANE

Results in the image being transferred to the graphics plane.

DIVE_BUFFER_ALTERNATE_PLANE

Results in the image being transferred to the alternate (for example, overlay) plane. If your hardware does not support such a plane, this is an error.

pbLineMask ([PBYTE](#)) - input

Indicates the line mask pointer for the line changed.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_BLITTER_NOT_SETUP

[DiveSetupBlitter](#) must be called before a call is made to DiveBlitImageLines.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveBlitImageLines - Remarks

When only a line changes, use DiveBlitImageLines to improve performance over blitting the entire image using [DiveBlitImage](#).

DiveBlitImageLines - Related Functions

- [DiveBlitImage](#)

DiveBlitImageLines - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

DiveCalcFrameBufferAddress

DiveCalcFrameBufferAddress - Syntax

This is a helper function that allows applications to calculate the linear frame-buffer address. This function can be used only if [DiveQueryCaps](#) indicates that DIVE is available and an *!NonScreenInstance* DIVE instance has been opened.

```
#include <dive.h>

HDIVE      hDiveInst;          /* Display engine DIVE instance. */
PRECTL     prectlDest;         /* Destination rectangle. */
PBYTE      *ppDestinationAddress; /* Calculated linear address. */
PULONG     pulBankNumber;      /* Bank number. */
PULONG     pulRemLinesInBank;   /* Number of lines in bank. */
ULONG      rc;                 /* Return codes. */

rc = DiveCalcFrameBufferAddress(hDiveInst,
                                prectlDest, ppDestinationAddress, pulBankNumber,
                                pulRemLinesInBank);
```

DiveCalcFrameBufferAddress Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveCalcFrameBufferAddress Parameter - prectlDest

prectlDest ([PRECTL](#)) - input
Destination rectangle in screen coordinates (0,0 bottom left). The *xLeft*, *yTop* point indicates the point for which the address is required. In addition, the width of this rectangle is used to calculate the remaining scan lines in this aperture (useful on bank-switched displays).

DiveCalcFrameBufferAddress Parameter - ppDestinationAddress

ppDestinationAddress ([PBYTE *](#)) - output
Indicates the calculated linear address within the frame buffer/aperture bank.

DiveCalcFrameBufferAddress Parameter - pulBankNumber

pulBankNumber ([PULONG](#)) - output
Indicates the bank number containing the top-left pel in the image.

DiveCalcFrameBufferAddress Parameter - pulRemLinesInBank

pulRemLinesInBank ([PULONG](#)) - output
Indicates the number of lines in the bank, beginning with the first line of the destination rectangle.

DiveCalcFrameBufferAddress Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS
If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_NO_DIRECT_ACCESS
The display adapter, display driver, or current video mode does not support direct-to-screen access.

DiveCalcFrameBufferAddress - Parameters

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

prectlDest ([PRECTL](#)) - input
Destination rectangle in screen coordinates (0,0 bottom left). The *xLeft*, *yTop* point indicates the point for which the address is required. In addition, the width of this rectangle is used to calculate the remaining scan lines in this aperture (useful on bank-switched displays).

ppDestinationAddress ([PBYTE *](#)) - output
Indicates the calculated linear address within the frame buffer/aperture bank.

pulBankNumber ([PULONG](#)) - output
Indicates the bank number containing the top-left pel in the image.

pulRemLinesInBank ([PULONG](#)) - output
Indicates the number of lines in the bank, beginning with the first line of the destination rectangle.

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS
If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_NO_DIRECT_ACCESS

The display adapter, display driver, or current video mode does not support direct-to-screen access.

DiveCalcFrameBufferAddress - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

DiveClose

DiveClose - Syntax

This function closes a display engine instance.

```
#include <dive.h>

HDIVE    hDiveInst; /* Instance identifier. */
ULONG    rc;        /* Return codes. */

rc = DiveClose(hDiveInst);
```

DiveClose Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input

Instance identifier. All data associated with this instance is freed.

DiveClose Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveClose - Parameters

hDiveInst ([HDIVE](#)) - input

Instance identifier. All data associated with this instance is freed.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveClose - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Glossary](#)

DiveDeacquireFrameBuffer

DiveDeacquireFrameBuffer - Syntax

This function releases exclusive access to the frame buffer.

```
#include <dive.h>
```

```
HDIVE    hDiveInst; /* Display engine DIVE instance. */
ULONG    rc;      /* Return codes. */
```

```
rc = DiveDeacquireFrameBuffer(hDiveInst);
```

DiveDeacquireFrameBuffer Parameter - hDiveInst

hDiveInst (**HDIVE**) - input
Display engine DIVE instance.

DiveDeacquireFrameBuffer Return Value - rc

rc (**ULONG**) - returns
Return codes indicating success or type of failure:

- DIVE_SUCCESS
If the function succeeds, 0 is returned.
- DIVE_ERR_DEACQUIRE_FAILED
The deacquire action did not complete successfully.
- DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.
- DIVE_ERR_NO_DIRECT_ACCESS
The display adapter, display driver, or current video mode does not support direct to screen access.

DiveDeacquireFrameBuffer - Parameters

hDiveInst (**HDIVE**) - input
Display engine DIVE instance.

rc (**ULONG**) - returns
Return codes indicating success or type of failure:

- DIVE_SUCCESS
If the function succeeds, 0 is returned.
- DIVE_ERR_DEACQUIRE_FAILED
The deacquire action did not complete successfully.
- DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.
- DIVE_ERR_NO_DIRECT_ACCESS
The display adapter, display driver, or current video mode does not support direct to screen access.

DiveDeacquireFrameBuffer - Remarks

This function returns an error if the frame buffer is not currently in the acquired state.

DiveDeacquireFrameBuffer - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

DiveEndImageBufferAccess

DiveEndImageBufferAccess - Syntax

This function must be called after reading or writing data in a buffer allocated with [DiveAllocImageBuffer](#) and corresponding to the [DiveBeginImageBufferAccess](#).

```
#include <dive.h>

HDIVE    hDiveInst;          /* Display engine DIVE instance. */
ULONG    ulBufferNumber;     /* Buffer number for which access is complete. */
ULONG    rc;                 /* Return codes. */

rc = DiveEndImageBufferAccess(hDiveInst, ulBufferNumber);
```

DiveEndImageBufferAccess Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveEndImageBufferAccess Parameter - ulBufferNumber

ulBufferNumber ([ULONG](#)) - input
DIVE instance buffer number for which access is complete.

DiveEndImageBufferAccess Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_BUFFER_NUMBER

The *ulBufferNumber* parameter must be a previously allocated or associated buffer number before it can be accessed.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveEndImageBufferAccess - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

ulBufferNumber ([ULONG](#)) - input

DIVE instance buffer number for which access is complete.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_BUFFER_NUMBER

The *ulBufferNumber* parameter must be a previously allocated or associated buffer number before it can be accessed.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveEndImageBufferAccess - Remarks

This function has no effect if the image buffer was allocated in system memory.

DiveEndImageBufferAccess - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

DiveFreeImageBuffer

DiveFreeImageBuffer - Syntax

This function deallocates an image buffer allocated by [DiveAllocImageBuffer](#).

```
#include <dive.h>

HDIVE    hDiveInst;        /* Display engine DIVE instance. */
ULONG    ulBufferNumber;   /* Buffer number to free. */
ULONG    rc;               /* Return codes. */

rc = DiveFreeImageBuffer(hDiveInst, ulBufferNumber);
```

DiveFreeImageBuffer Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveFreeImageBuffer Parameter - ulBufferNumber

ulBufferNumber ([ULONG](#)) - input
Buffer number to free in this instance.

DiveFreeImageBuffer Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

- DIVE_SUCCESS
If the function succeeds, 0 is returned.
- DIVE_ERR_INVALID_BUFFER_NUMBER
The *ulBufferNumber* parameter must be a previously allocated or associated buffer number.
- DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveFreeImageBuffer - Parameters

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

ulBufferNumber ([ULONG](#)) - input
Buffer number to free in this instance.

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS
If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_BUFFER_NUMBER
The *ulBufferNumber* parameter must be a previously allocated or associated buffer number.

DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveFreeImageBuffer - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Glossary](#)

DiveOpen

DiveOpen - Syntax

This function opens a display engine instance.

```
#include <dive.h>

HDIVE    *phDiveInst;          /* Display engine instance. */
BOOL     fNonScreenInstance;   /* Using physical display hardware. */
PVOID     ppFrameBuffer;       /* Linear address of the VRAM. */
ULONG     rc;                  /* Return codes. */

rc = DiveOpen(phDiveInst, fNonScreenInstance,
              ppFrameBuffer);
```

DiveOpen Parameter - phDiveInst

phDiveInst ([HDIVE *](#)) - output

Upon successful open of a DIVE instance, this parameter points to the display engine instance. If there is an error, this parameter is set to NULL and an error is returned as the return code.

DiveOpen Parameter - fNonScreenInstance

fNonScreenInstance (BOOL) - input

This parameter indicates whether this DIVE instance should be using the physical display hardware. If *fNonScreenInstance* is FALSE, this function returns a pointer to the linear base address to the VRAM and the [DiveBlitImage](#) functions in this instance can have screen or non-screen destination buffers. If *fNonScreenInstance* is TRUE, the DIVE instance will not use display hardware resources, and only non-screen destination buffers can be used as destination buffers for [DiveBlitImage](#).

DiveOpen Parameter - ppFrameBuffer

ppFrameBuffer (PVOID) - output

This parameter is a pointer on input to a pointer variable that will be set on output to contain the linear address of the VRAM if *fNonScreenInstance* is specified. This address always corresponds to the top left of the screen when aperture bank 0 is selected. If you do *not* intend to use this, pass a 0.

DiveOpen Return Value - rc

rc (ULONG) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_ALLOCATION_ERROR

The hardware-blitter memory allocation failed.

DIVE_ERR_NO_DIRECT_ACCESS

The display adapter does not support direct access, the display is in 16-color mode, or the display driver is down-level.

DIVE_ERR_SSMDD_NOT_INSTALLED

The device driver SSMDD.SYS is missing from CONFIG.SYS.

DIVE_ERR_TOO_MANY_INSTANCES

There are not enough resources for another DIVE instance.

DiveOpen - Parameters

phDiveInst (HDIVE *) - output

Upon successful open of a DIVE instance, this parameter points to the display engine instance. If there is an error, this parameter is set to NULL and an error is returned as the return code.

fNonScreenInstance ([BOOL](#)) - input

This parameter indicates whether this DIVE instance should be using the physical display hardware. If *fNonScreenInstance* is FALSE, this function returns a pointer to the linear base address to the VRAM and the [DiveBlitImage](#) functions in this instance can have screen or non-screen destination buffers. If *fNonScreenInstance* is TRUE, the DIVE instance will not use display hardware resources, and only non-screen destination buffers can be used as destination buffers for [DiveBlitImage](#).

ppFrameBuffer ([PVOID](#)) - output

This parameter is a pointer on input to a pointer variable that will be set on output to contain the linear address of the VRAM if *fNonScreenInstance* is specified. This address always corresponds to the top left of the screen when aperture bank 0 is selected. If you do *not* intend to use this, pass a 0.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_ALLOCATION_ERROR

The hardware-blitter memory allocation failed.

DIVE_ERR_NO_DIRECT_ACCESS

The display adapter does not support direct access, the display is in 16-color mode, or the display driver is down-level.

DIVE_ERR_SSMDD_NOT_INSTALLED

The device driver SSMDD.SYS is missing from CONFIG.SYS.

DIVE_ERR_TOO_MANY_INSTANCES

There are not enough resources for another DIVE instance.

DiveOpen - Remarks

If the *fNonScreenInstance* parameter is FALSE, DIVE will return the linear base address of the VRAM.

DiveOpen does not require direct addressability to the screen because the DIVE display engine is also intended as a color-format conversion tool.

DiveOpen - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

DiveQueryCaps

DiveQueryCaps - Syntax

This function returns static information about the characteristics and capabilities of the display hardware and available driver functions. This function can be called without performing a [DiveOpen](#) call.

```
#include <dive.h>

PDIVE_CAPS    pDiveCaps;      /* Pointer to data structure. */
ULONG         ulPlaneBufNum;  /* Indicates plane. */
ULONG         rc;             /* Return codes. */

rc = DiveQueryCaps(pDiveCaps, ulPlaneBufNum);
```

DiveQueryCaps Parameter - pDiveCaps

pDiveCaps ([PDIVE_CAPS](#)) - input
Pointer to [DIVE_CAPS](#). This structure defines the color-conversion capabilities of the DIVE component.

DiveQueryCaps Parameter - ulPlaneBufNum

ulPlaneBufNum ([ULONG](#)) - input
Indicates the plane for which information is to be returned.

[DIVE_BUFFER_SCREEN](#)
Return information for graphics plane. This is the default.

[DIVE_BUFFER_GRAPHICS_PLANE](#)
Return information for graphics plane.

[DIVE_BUFFER_ALTERNATE_PLANE](#)
Return information for overlay plane, if present.

DiveQueryCaps Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

[DIVE_SUCCESS](#)
If the function succeeds, 0 is returned.

[DIVE_ERR_INSUFFICIENT_LENGTH](#)
The *pFormatData* of length *ulFormatLength* (specified in the [DIVE_CAPS](#) structure) is not large enough for the total number of input and output formats.

DiveQueryCaps - Parameters

pDiveCaps ([PDIVE_CAPS](#)) - input
Pointer to [DIVE_CAPS](#). This structure defines the color-conversion capabilities of the DIVE component.

ulPlaneBufNum ([ULONG](#)) - input
Indicates the plane for which information is to be returned.

DIVE_BUFFER_SCREEN
Return information for graphics plane. This is the default.

DIVE_BUFFER_GRAPHICS_PLANE
Return information for graphics plane.

DIVE_BUFFER_ALTERNATE_PLANE
Return information for overlay plane, if present.

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS
If the function succeeds, 0 is returned.

DIVE_ERR_INSUFFICIENT_LENGTH
The *pFormatData* of length *ulFormatLength* (specified in the [DIVE_CAPS](#) structure) is not large enough for the total number of input and output formats.

DiveQueryCaps - Remarks

This function indicates whether direct access through DIVE is possible. A list of input and output color formats supported by the blitter is returned. Some of these might be assisted through hardware color conversion; however, this detail is hidden from the user as the display engine will provide hardware assistance, if present.

If the length of the format buffer specified in *ulFormatLength* of [DIVE_CAPS](#) is insufficient to contain all of the supported input and output formats, DIVE_ERR_INSUFFICIENT_LENGTH is returned, and the field is updated with the required length.

DiveQueryCaps - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DiveSetDestinationPalette

DiveSetDestinationPalette - Syntax

This function sets the palette associated with the destination of [DiveBlitImage](#). This capability is useful for outputting direct color image formats on 8-bit (256) displays. If no destination palette has been set, direct color space to LUT8 color-space conversion will be performed assuming the destination palette is the standard OS/2 palette. If DIVE is being used to transfer direct color images to an 8-bit display and the physical palette changes, it is expected that the application will provide notification of the change to the DIVE instance using this function.

```
#include <dive.h>

HDIVE    hDiveInst;    /* Display engine DIVE instance. */
ULONG    ulStartIndex; /* Index of first palette entry to set. */
ULONG    ulNumEntries; /* Number of palette entries to set. */
PBYTE    pbRGB2Entries; /* Palette for target image space. */
ULONG    rc;           /* Return codes. */

rc = DiveSetDestinationPalette(hDiveInst,
                               ulStartIndex, ulNumEntries, pbRGB2Entries);
```

DiveSetDestinationPalette Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveSetDestinationPalette Parameter - ulStartIndex

ulStartIndex ([ULONG](#)) - input
Starting location of the palette changes; usually 0.

DiveSetDestinationPalette Parameter - ulNumEntries

ulNumEntries ([ULONG](#)) - input
Number of entries of palette change; usually 256.

DiveSetDestinationPalette Parameter - pbRGB2Entries

pbRGB2Entries ([PBYTE](#)) - input
Indicates the palette for the target image space. This does not set the physical palette.

DiveSetDestinationPalette Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_INVALID_PALETTE

The palette specified for the destination data is invalid.

DiveSetDestinationPalette - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

ulStartIndex ([ULONG](#)) - input

Starting location of the palette changes; usually 0.

ulNumEntries ([ULONG](#)) - input

Number of entries of palette change; usually 256.

pbRGB2Entries ([PBYTE](#)) - input

Indicates the palette for the target image space. This does not set the physical palette.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_INVALID_PALETTE

The palette specified for the destination data is invalid.

DiveSetDestinationPalette - Remarks

Neither [DiveSetSourcePalette](#) nor [DiveSetDestinationPalette](#) will set the physical palette. If your application needs to set the physical palette (that is, all 256 entries), it must do so as a full-screen (maximized) application. No WM_REALIZEPALETTE message will be sent to other applications, and no redraw is done.

If [DiveSetDestinationPalette](#) is not called and the display resolution is LUT8, then the current physical palette will be used. If [DiveSetDestinationPalette](#) is not called and the display resolution is *not* LUT8, then the OS/2 default palette will be used.

Most applications will notify DIVE of a physical palette change when a WM_REALIZEPALETTE message comes in on the message queue. You need not query the palette in this scenario, as a call to [DiveSetDestinationPalette](#)(hDiveInst, 0, 256, 0) will query the physical palette for the application.

DiveSetDestinationPalette - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

DiveSetSourcePalette

DiveSetSourcePalette - Syntax

This function sets the palette associated with the source data to be used for translating from 8-bit (LUT8) format to other color formats, for example, RGB 5-6-5 or RGB24.

```
#include <dive.h>

HDIVE    hDiveInst;      /* Display engine DIVE instance. */
ULONG    ulStartIndex;   /* First palette entry to set. */
ULONG    ulNumEntries;   /* Number of palette entries to set. */
PBYTE    pbRGB2Entries;  /* RGB values to be set. */
ULONG    rc;             /* Return codes. */

rc = DiveSetSourcePalette(hDiveInst, ulStartIndex,
                          ulNumEntries, pbRGB2Entries);
```

DiveSetSourcePalette Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveSetSourcePalette Parameter - ulStartIndex

ulStartIndex ([ULONG](#)) - input
Index of first palette entry to set.

DiveSetSourcePalette Parameter - ulNumEntries

ulNumEntries ([ULONG](#)) - input

Number of palette entries to set. If this value is zero, the source palette is "reset" and DiveBlitImage will no longer perform palette translations when both source and destination formats are LUT8.

DiveSetSourcePalette Parameter - pbRGB2Entries

pbRGB2Entries ([PBYTE](#)) - input

Indicates number of RGB values to be set in the source palette for use when blitting from LUT8 source format.

DiveSetSourcePalette Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_INVALID_PALETTE

The palette specified for the source data is invalid.

DiveSetSourcePalette - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

ulStartIndex ([ULONG](#)) - input

Index of first palette entry to set.

ulNumEntries ([ULONG](#)) - input

Number of palette entries to set. If this value is zero, the source palette is "reset" and DiveBlitImage will no longer perform palette translations when both source and destination formats are LUT8.

pbRGB2Entries ([PBYTE](#)) - input

Indicates number of RGB values to be set in the source palette for use when blitting from LUT8 source format.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_INVALID_PALETTE

The palette specified for the source data is invalid.

This function is intended for use when the source data is palettized.

This function is useful for displaying LUT8 source images from applications that perform palette animation. If no source palette is specified, the OS/2 default palette values will be used for translating LUT8 source data to direct color formats. If the source and destination color formats are both LUT8 and no source palette or no destination palette has been set, [DiveBltImage](#) will transfer the 8-bit image without performing any translation. If the source and destination color formats are both LUT8 and both palettes have been set, [DiveBltImage](#) will perform palette translation between the source palette (associated with the source image) and the destination palette (usually the physical palette, in the case where the destination is the screen).

If DiveSetSourcePalette is not called and the source format is LUT8, then the current physical palette will be used. If DiveSetSourcePalette is not called and the source format is *not* LUT8, then the OS/2 default palette will be used.

Neither `DiveSetSourcePalette` nor `DiveSetDestinationPalette` will set the physical palette. If your application needs to set the physical palette (that is, all 256 entries), it must do so as a full-screen (maximized) application. No `WM_REALIZEPALETTE` message will be sent to other applications, and no redraw is done.

Select an item:

Syntax

Parameters

Returns

Remarks

Glossary

This function enables transparent blitting using a chroma-key color or chroma-keying on a color space range. `DiveSetTransparentBlitMode` must be called *before* calling [DiveSetupBlitter](#).

```
#include <dive.h>
```

```

HDIVE      hDiveInst;          /* Display engine DIVE instance. */
ULONG      ulTransBlitMode;    /* Transparent blit mode. */
ULONG      ulValue1;           /* Minimum value in range. */
ULONG      ulValue2;           /* Maximum value in range. */
ULONG      rc;                 /* Return codes. */

```

[illegible]

DiveSetTransparentBlitMode Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveSetTransparentBlitMode Parameter - ulTransBlitMode

ulTransBlitMode ([ULONG](#)) - input
Specifies the mode for transparent blitting. Set *ulTransBlitMode* to one of the following values:

DIVE_TBM_NONE	No transparency; that is, all pixels are transferred. This is the default.
DIVE_TBM_EXCLUDE_SOURCE_VALUE	Source pixels with values that exactly match the value specified in <i>ulValue1</i> are not transferred.
DIVE_TBM_EXCLUDE_SOURCE_RGB_RANGE	Source pixels with values that are within the range specified in the RGB color space defined by <i>ulValue1</i> (minimum) and <i>ulValue2</i> (maximum) are not transferred by DiveBlitImage .
DIVE_TBM_INCLUDE_SOURCE_RGB_RANGE	Source pixels with values that are outside the range specified in the RGB color space defined by <i>ulValue1</i> (minimum) and <i>ulValue2</i> (maximum) are not transferred by DiveBlitImage .
DIVE_TBM_EXCLUDE_SOURCE_YUV_RANGE	Source pixels with values that are outside the range specified in the YUV color space defined by <i>ulValue1</i> (minimum) and <i>ulValue2</i> (maximum) are not transferred by DiveBlitImage .
DIVE_TBM_INCLUDE_SOURCE_YUV_RANGE	Source pixels with values that are outside the range specified in the YUV color space defined by <i>ulValue1</i> (minimum) and <i>ulValue2</i> (maximum) are not transferred by DiveBlitImage .

DiveSetTransparentBlitMode Parameter - ulValue1

ulValue1 ([ULONG](#)) - input
Minimum value in range.

DiveSetTransparentBlitMode Parameter - ulValue2

ulValue2 ([ULONG](#)) - input
Maximum value in range.

DiveSetTransparentBlitMode Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveSetTransparentBlitMode - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

uiTransBlitMode ([ULONG](#)) - input

Specifies the mode for transparent blitting. Set *uiTransBlitMode* to one of the following values:

DIVE_TBM_NONE

No transparency; that is, all pixels are transferred. This is the default.

DIVE_TBM_EXCLUDE_SOURCE_VALUE

Source pixels with values that exactly match the value specified in *uiValue1* are not transferred.

DIVE_TBM_EXCLUDE_SOURCE_RGB_RANGE

Source pixels with values that are within the range specified in the RGB color space defined by *uiValue1* (minimum) and *uiValue2* (maximum) are not transferred by [DiveBlitImage](#).

DIVE_TBM_INCLUDE_SOURCE_RGB_RANGE

Source pixels with values that are outside the range specified in the RGB color space defined by *uiValue1* (minimum) and *uiValue2* (maximum) are not transferred by [DiveBlitImage](#).

DIVE_TBM_EXCLUDE_SOURCE_YUV_RANGE

Source pixels with values that are outside the range specified in the YUV color space defined by *uiValue1* (minimum) and *uiValue2* (maximum) are not transferred by [DiveBlitImage](#).

DIVE_TBM_INCLUDE_SOURCE_YUV_RANGE

Source pixels with values that are outside the range specified in the YUV color space defined by *uiValue1* (minimum) and *uiValue2* (maximum) are not transferred by [DiveBlitImage](#).

uiValue1 ([ULONG](#)) - input

Minimum value in range.

uiValue2 ([ULONG](#)) - input

Maximum value in range.

rc ([ULONG](#)) - returns

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DiveSetTransparentBlitMode - Remarks

The transparent blitting function is based on source pixel values. A pixel in the destination image buffer is not modified if the corresponding pixel in the source buffer is "transparent". The interpretation of the color values specified in *uiValue1* and *uiValue2* is dependent on the source image color format (*fccSrcColorFormat* in the [SETUP_BLITTER](#) structure) and the transparent blitting mode.

The following table describes the color and range specification:

FOURCC_LUT8	The color value is specified in the low 8 bits of the parameter.
FOURCC_Y888, FOURCC_Y2X2, FOURCC_Y4X4, FOURCC_YUV9, FOURCC_Y644, FOURCC_Y422	23:8 - Y, 15:8 - U, 7:8 - V (bits 31:8 ignored)
FOURCC_R565, FOURCC_R555, FOURCC_R664, FOURCC_RGB3, FOURCC_BGR3, FOURCC_RGB4, FOURCC_BGR4	23:8 - R, 15:8 - G, 7:8 - B (bits 31:8 ignored). R, G, and B components are specified with 8-bit significance regardless of significance in source data.

Transparent blitting of other source image formats is not supported.

For range comparisons in RGB or YUV, the three components are compared independently against the minimum and maximum values specified by the *uiValue1* and *uiValue2* parameters, respectively. A value is considered to be within the specified range if it is greater to or equal to *uiValue1* and less than or equal to *uiValue2*.

For DIVE_TBM_EXCLUDE_SOURCE_VALUE transparent blitting, the specified value in *uiValue1* is assumed to be in the source color space as described above. For range comparisons, the values specified in *uiValue1* and *uiValue2* are assumed to be in the color space in which the range comparison is to be performed, either YUV or RGB.

With LUT8 source format, in DIVE_TBM_EXCLUDE_SOURCE_VALUE transparent blitting mode, the value is assumed to be the original LUT8 value. For other transparent blitting modes (RGB or YUV range), an LUT8 value is converted to a direct color value based on the current source palette to determine whether or not it is within the specified range.

DiveSetTransparentBlitMode - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

DiveSetupBlitter

DiveSetupBlitter - Syntax

This function sets up blitter operations.

```
#include <dive.h>

HDIVE      hDiveInst;      /* Display engine DIVE instance. */
PSETUP_BLITTER pSetupBlitter; /* Pointer to SETUP_BLITTER. */
ULONG      rc;             /* Return codes. */

rc = DiveSetupBlitter(hDiveInst, pSetupBlitter);
```

DiveSetupBlitter Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input
Display engine DIVE instance.

DiveSetupBlitter Parameter - pSetupBlitter

pSetupBlitter ([PSETUP_BLITTER](#)) - input
Pointer to [SETUP_BLITTER](#). This data structure contains parameters that will be used during the subsequent [DiveBlitImage](#) call. If this pointer is zero on input, this function will deinitialize the blitter and subsequent calls to [DiveBlitImage](#) will return [DIVE_ERR_BLITTER_NOT_SETUP](#).

The *ulStructLen* field specifies the number of fields that are valid on input. Fields that are not specified as valid on input are considered to be unchanged from previous calls to [DiveSetupBlitter](#). For example, if the visible rectangles have not changed, the structure length can be set to exclude the *ulNumDstRects* and *pVisDstRects* fields of [SETUP_BLITTER](#), thereby enabling a performance optimization in the function.

DiveSetupBlitter Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or type of failure:

- [DIVE_SUCCESS](#)
If the function succeeds, 0 is returned.
- [DIVE_ERR_INVALID_CONVERSION](#)
The source image format cannot be converted as requested to the destination image format.
- [DIVE_ERR_INVALID_INSTANCE](#)
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.
- [DIVE_ERR_SOURCE_FORMAT](#)
The source format is not a recognized FOURCC.
- [DIVE_WARN_NO_SIZE](#)
No blitting will occur because either the destination width or height is zero.

DiveSetupBlitter - Parameters

hDiveInst (**HDIVE**) - input
Display engine DIVE instance.

pSetupBlitter (**PSETUP_BLITTER**) - input
Pointer to **SETUP_BLITTER**. This data structure contains parameters that will be used during the subsequent **DiveBlitImage** call. If this pointer is zero on input, this function will deinitialize the blitter and subsequent calls to **DiveBlitImage** will return **DIVE_ERR_BLITTER_NOT_SETUP**.

The *ulStructLen* field specifies the number of fields that are valid on input. Fields that are not specified as valid on input are considered to be unchanged from previous calls to **DiveSetupBlitter**. For example, if the visible rectangles have not changed, the structure length can be set to exclude the *ulNumDstRects* and *pVisDstRects* fields of **SETUP_BLITTER**, thereby enabling a performance optimization in the function.

rc (**ULONG**) - returns
Return codes indicating success or type of failure:

DIVE_SUCCESS
If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_CONVERSION
The source image format cannot be converted as requested to the destination image format.

DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_SOURCE_FORMAT
The source format is not a recognized FOURCC.

DIVE_WARN_NO_SIZE
No blitting will occur because either the destination width or height is zero.

DiveSetupBlitter - Remarks

The visible rectangles are passed using the *pVisDstRects* parameter of **SETUP_BLITTER**. These rectangles are relative to the destination window origin. The display engine will determine if it is fully visible if *ulNumDstRects* = 1 and the first **pVisDstRects* = {0, 0, width, height}. The display engine will also assume the image is fully clipped if *ulNumDstRects* = 0 or if *ulNumDstRects* = 1 and the rectangle width or height are zero.

If **DiveSetupBlitter** determines that the requested blitter operations to follow can be performed partially or wholly using display hardware features, it will do so. If the requested input format is supported by the software emulation within the display engine but not by the display hardware, **DiveSetupBlitter** will enable conversion to a format that is supported by the hardware to an internally allocated buffer. The converted image will, in turn, be transferred from the internal buffer to the screen by the hardware.

When PM sends a **WM_VRNDISABLED** message, the application must inform DIVE that the visible regions are about to enter a changing state. This is done with a call to **DiveSetupBlitter** (*hDive*, 0). When the state of visible regions again becomes static, PM sends a **WM_VRNENABLED** message and a normal call to **DiveSetupBlitter** is used.

DiveSetupBlitter was not intended to be called at high frequency. If several different source/destination size pairs are to be used, try opening several instances to increase performance.

Note: The *fNonScreenInstance* parameter specified on **DiveOpen** indicates whether or not **DiveSetupBlitter** should use acceleration hardware when present.

DiveSetupBlitter - Related Functions

- [DiveBlitImage](#)

DiveSetupBlitter - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Related Functions](#)

[Glossary](#)

DiveSwitchBank

DiveSwitchBank - Syntax

This function selects the VRAM bank for bank-switched displays.

```
#include <dive.h>
```

```
HDIVE    hDiveInst;    /* Display engine DIVE instance. */
ULONG    ulBankNumber; /* Bank index to switch to. */
ULONG    rc;           /* Return codes. */
```

```
rc = DiveSwitchBank(hDiveInst, ulBankNumber);
```

DiveSwitchBank Parameter - hDiveInst

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

DiveSwitchBank Parameter - ulBankNumber

ulBankNumber ([ULONG](#)) - input

Bank index to switch to.

DiveSwitchBank Return Value - rc

rc ([ULONG](#)) - returns

This function returns an error return code if any of the following conditions are met:

- Direct-screen access through the DIVE or EnDIVE dev_escapes are unavailable.
- The current *hDiveInst* has not issued [DiveAcquireFrameBuffer](#).
- The frame buffer consists of a single aperture.

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_BANK_SWITCH_FAILED

The bank could not be switched to the specified parameters.

DIVE_ERR_NO_DIRECT_ACCESS

The display adapter, display driver, or current video mode does not support direct-to-screen access.

DiveSwitchBank - Parameters

hDiveInst ([HDIVE](#)) - input

Display engine DIVE instance.

ulBankNumber ([ULONG](#)) - input

Bank index to switch to.

rc ([ULONG](#)) - returns

This function returns an error return code if any of the following conditions are met:

- Direct-screen access through the DIVE or EnDIVE dev_escapes are unavailable.
- The current *hDiveInst* has not issued [DiveAcquireFrameBuffer](#).
- The frame buffer consists of a single aperture.

Return codes indicating success or type of failure:

DIVE_SUCCESS

If the function succeeds, 0 is returned.

DIVE_ERR_INVALID_INSTANCE

The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

DIVE_ERR_BANK_SWITCH_FAILED

The bank could not be switched to the specified parameters.

DIVE_ERR_NO_DIRECT_ACCESS

The display adapter, display driver, or current video mode does not support direct-to-screen access.

DiveSwitchBank - Remarks

Applications will not need to use this function if the frame buffer contains only a single aperture. The `fMultipleAperture` flag returned from [DiveQueryCaps](#) determines if this is necessary.

DiveSwitchBank - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)

Real-Time MIDI Functions

The real-time MIDI subsystem (RTMIDI) provides support for handling real-time MIDI operations for applications. The following table describes the RTMIDI functions that are currently supported.

All of the RTMIDI functions have `MIDIERR_INTERNAL_SYSTEM` and `MIDIERR_INVALID_PARAMETER` as possible returns (with the exception of [TimerGetPointer](#) and [TimerSleep](#), which have `TIMERERR_INTERNAL_SYSTEM` and `TIMERERR_INVALID_PARAMETER` as possible returns). If `MIDIERR_INTERNAL_SYSTEM` or `TIMERERR_INTERNAL_SYSTEM` is returned, the application should shut down immediately and not attempt any further calls to RTMIDI.

Function	Description
MIDIAddLink	Creates a link from the source instance to a target instance.
MIDICreateInstance	Creates an instance.
MIDIDeleteInstance	Deletes an instance.
MIDIDisableInstance	Disables an instance from sending or receiving messages.
MIDIEnableInstance	Enables an instance for sending or receiving messages.
MIDIQueryClassList	Returns the names of all valid current classes.
MIDIQueryInstanceList	Returns a sequential list of relevant instance data for all instances in the chain.
MIDIQueryInstanceName	Returns the name of an instance.
MIDIQueryNumClasses	Returns the number of classes currently registered.
MIDIQueryNumInstances	Returns the number of instances.
MIDIQueryVersion	Returns the current RTMIDI device driver and DLL version numbers.
MIDIRemoveLink	Removes a link from the source instance to a target instance.
MIDISendMessages	Sends a block of MIDI messages.
MIDISetup	Provides a generic setup for an application.

<code>MIDITimer</code>	Starts or stops the RTMIDI timer.
<code>TimerGetPointer</code>	Obtains a pointer to the RTMIDI timer.
<code>TimerSleep</code>	Sets the duration in milliseconds to trigger an event originated by the application.

MIDIAddLink

MIDIAddLink - Syntax

This function creates a link from the source instance (*minstanceSource*) to the target instance (*minstanceTarget*).

```
#include <mididll.h>

MINSTANCE minstanceSource; /* Source. */
MINSTANCE minstanceTarget; /* Target. */
ULONG ulSlotNumber; /* Slot number. */
ULONG ulFlag; /* Not used. */
ULONG rc; /* Return code. */

rc = MIDIAddLink(minstanceSource, minstanceTarget,
    ulSlotNumber, ulFlag);
```

MIDIAddLink Parameter - minstanceSource

minstanceSource ([MINSTANCE](#)) - input
The source instance number.

MIDIAddLink Parameter - minstanceTarget

minstanceTarget ([MINSTANCE](#)) - input
The target instance number.

MIDIAddLink Parameter - ulSlotNumber

ulSlotNumber ([ULONG](#)) - input
The slot number of the source instance.

MIDIAddLink Parameter - ulFlag

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

MIDIAddLink Return Value - rc

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_INSTANCE_NUMBER
Either the source or target instance number is invalid.

MIDIERR_INVALID_PARAMETER
The *ulSlotNumber* parameter contains an invalid value.

MIDIERR_NOT_ALLOWED
An attempt was made to create a link either between two instances not owned by this process or between a hardware node and an instance owned by another process.

MIDIERR_RESOURCE_NOT_AVAILABLE
There are not enough resources to create any new links.

MIDIAddLink - Parameters

minstanceSource ([MINSTANCE](#)) - input
The source instance number.

minstanceTarget ([MINSTANCE](#)) - input
The target instance number.

ulSlotNumber ([ULONG](#)) - input
The slot number of the source instance.

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_INSTANCE_NUMBER
Either the source or target instance number is invalid.

MIDIERR_INVALID_PARAMETER
The *ulSlotNumber* parameter contains an invalid value.

MIDIERR_NOT_ALLOWED
An attempt was made to create a link either between two instances not owned by this process or between a hardware node and an instance owned by another process.

MIDIERR_RESOURCE_NOT_AVAILABLE
There are not enough resources to create any new links.

MIDIAddLink - Remarks

A link is only allowed between two hardware nodes, a hardware node and an instance owned by this process, or between two instances which are owned by this process.

MIDIAddLink - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDICreateInstance

MIDICreateInstance - Syntax

This function creates an instance of the specified class.

```
#include <mididl1.h>

ULONG      ulClassNumber;    /* Class number. */
PMINSTANCE pminstance;      /* New instance #. */
PSZ        pszInstanceName; /* Instance name. */
ULONG      ulFlag;          /* Not used. */
ULONG      rc;               /* Return code. */
```

```
rc = MIDICreateInstance(ulClassNumber, pminstance,
                        pszInstanceName, ulFlag);
```

MIDICreateInstance Parameter - ulClassNumber

ulClassNumber (ULONG) - input
The class number.

MIDICreateInstance Parameter - pminstance

pminstance (PINSTANCE) - output
The number of the new instance.

MIDICreateInstance Parameter - pszInstanceName

pszInstanceName (PSZ) - input
Pointer to a string of length MIDI_NAME_LENGTH containing the instance name.

MIDICreateInstance Parameter - ulFlag

ulFlag (ULONG) - input
This parameter is not currently used and must be set to 0.

MIDICreateInstance Return Value - rc

- rc** (LONG) - returns
Returns 0 if the command is completed successfully.
- MIDIERR_DUPLICATE_INSTANCE
The instance name already exists.
 - MIDIERR_INTERNAL_SYSTEM
System problem.
 - MIDIERR_INVALID_CLASS_NUMBER
The class number (*ulClassNumber*) is invalid.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_INSTANCE_NAME
The instance name (*pszInstanceName*) is invalid.

MIDIERR_INVALID_PARAMETER
The *pminstance* pointer is invalid.

MIDIERR_NOT_ALLOWED
An attempt was made to create a second application node.

MIDIERR_RESOURCE_NOT_AVAILABLE
A required system resource is not available.

MIDICreateInstance - Parameters

ulClassNumber (ULONG) - input
The class number.

pminstance (PMINSTANCE) - output
The number of the new instance.

pszInstanceName (PSZ) - input
Pointer to a string of length MIDI_NAME_LENGTH containing the instance name.

ulFlag (ULONG) - input
This parameter is not currently used and must be set to 0.

rc (ULONG) - returns
Returns 0 if the command is completed successfully.

MIDIERR_DUPLICATE_INSTANCE
The instance name already exists.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_CLASS_NUMBER
The class number (*ulClassNumber*) is invalid.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_INSTANCE_NAME
The instance name (*pszInstanceName*) is invalid.

MIDIERR_INVALID_PARAMETER
The *pminstance* pointer is invalid.

MIDIERR_NOT_ALLOWED
An attempt was made to create a second application node.

MIDIERR_RESOURCE_NOT_AVAILABLE
A required system resource is not available.

MIDICreateInstance - Remarks

Instances of type hardware class (hardware nodes) cannot be created.

See the *Multimedia Subsystem Programming Guide* for a description of pre-defined class names.

The process creating the instance is considered the instance owner. Ownership affects other operations. For example, a process cannot delete an instance belonging to another process. See [MIDIAddLink](#), [MIDIDeleteInstance](#), and [MIDIRemoveLink](#) for specific information.

MIDICreateInstance - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDIDeleteInstance

MIDIDeleteInstance - Syntax

This function deletes an instance.

```
#include <mididll.h>

MINSTANCE    minstance; /* Instance number. */
ULONG        ulFlag;    /* Not used. */
ULONG        rc;        /* Return code. */

rc = MIDIDeleteInstance(minstance, ulFlag);
```

MIDIDeleteInstance Parameter - minstance

minstance ([MINSTANCE](#)) - input
The instance to be deleted.

MIDIDeleteInstance Parameter - ulFlag

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

MIDIDeleteInstance Return Value - rc

rc (**ULONG**) - returns

Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_CLASS_NUMBER
An attempt was made to delete a hardware node.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_INSTANCE_NUMBER
The instance number (*minstance*) is invalid.

MIDIDeleteInstance - Parameters

minstance (**MINSTANCE**) - input

The instance to be deleted.

ulFlag (**ULONG**) - input

This parameter is not currently used and must be set to 0.

rc (**ULONG**) - returns

Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_CLASS_NUMBER
An attempt was made to delete a hardware node.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_INSTANCE_NUMBER
The instance number (*minstance*) is invalid.

MIDIDeleteInstance - Remarks

MIDIDeleteInstance also deletes links to and from the deleted instance.

Instances of type hardware class (hardware nodes) cannot be deleted.

Instances belonging to other processes cannot be deleted.

MIDIDeleteInstance - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDIDisableInstance

MIDIDisableInstance - Syntax

This function disables an instance from sending messages, receiving messages, or both.

```
#include <mididll.h>

MINSTANCE    minstance; /* Instance number. */
ULONG        ulFlag;    /* Flag. */
ULONG        rc;        /* Return code. */

rc = MIDIDisableInstance(minstance, ulFlag);
```

MIDIDisableInstance Parameter - minstance

minstance ([MINSTANCE](#)) - input
The instance to be disabled.

MIDIDisableInstance Parameter - ulFlag

ulFlag ([ULONG](#)) - input
Flag indicating the hardware node should be disabled from sending or receiving messages. Valid values are MIDI_DISABLE_SEND and MIDI_DISABLE_RECEIVE.

MIDIDisableInstance Return Value - rc

rc ([ULONG](#)) - returns

Returns 0 if the command is completed successfully.

MIDIERR_HARDWARE_FAILED

The disable or close attempt failed due to a hardware problem.

MIDIERR_INTERNAL_SYSTEM

System problem.

MIDIERR_INVALID_FLAG

The *ulFlag* value is invalid.

MIDIERR_INVALID_INSTANCE_NUMBER

The instance number (*minstance*) is invalid.

MIDIERR_NOT_ALLOWED

Both send and receive are not permitted for this particular instance.

MIDIERR_RECEIVEONLY

An attempt was made to disable an instance for send (or for send and receive) when the instance supports receive only.

MIDIERR_SENDOONLY

An attempt was made to disable an instance for receive (or for send and receive) when the instance supports send only.

MIDIDisableInstance - Parameters

minstance (**MINSTANCE**) - input

The instance to be disabled.

ulFlag (**ULONG**) - input

Flag indicating the hardware node should be disabled from sending or receiving messages. Valid values are MIDI_DISABLE_SEND and MIDI_DISABLE_RECEIVE.

rc (**ULONG**) - returns

Returns 0 if the command is completed successfully.

MIDIERR_HARDWARE_FAILED

The disable or close attempt failed due to a hardware problem.

MIDIERR_INTERNAL_SYSTEM

System problem.

MIDIERR_INVALID_FLAG

The *ulFlag* value is invalid.

MIDIERR_INVALID_INSTANCE_NUMBER

The instance number (*minstance*) is invalid.

MIDIERR_NOT_ALLOWED

Both send and receive are not permitted for this particular instance.

MIDIERR_RECEIVEONLY

An attempt was made to disable an instance for send (or for send and receive) when the instance supports receive only.

MIDIERR_SENDOONLY

An attempt was made to disable an instance for receive (or for send and receive) when the instance supports send only.

MIDIDisableInstance - Remarks

When an instance is disabled for a particular direction, the reference count for that direction is decremented. If this count reaches 0, the instance becomes truly disabled for that direction.

If the request is for both send and receive, and the instance only supports one direction, then either MIDIERR_SENDOONLY or MIDIERR_RECEIVEONLY is returned (as appropriate), and the instance is not disabled for either direction. If the instance supports neither send nor receive, MIDIERR_NOT_ALLOWED will be returned. Therefore, the application should call [MIDIQueryInstanceList](#) to determine the capabilities of the instance before attempting to disable it.

MIDIDisableInstance has special meaning for hardware nodes. When a hardware node becomes truly disabled, the Type A driver that it represents is issued an close request. If this request fails then MIDIERR_HARDWARE_FAILED is returned, and the instance remains enabled with a reference count of 1.

If an application attempts to disable an instance for both directions simultaneously, it is possible for one direction to succeed and for the other to fail. In this case, the instance will be disabled for only one direction and the application will not know which direction that is. This can happen when disabling a hardware node or when the instance was enabled for only one direction. In the former case, MIDIERR_HARDWARE_FAILED is returned and in the latter case, MIDIERR_NOT_ALLOWED is returned.

Therefore, it is suggested that an application make separate MIDIDisableInstance calls to disable the instance for both directions. If the first call fails, the application can then take appropriate measures.

MIDIDisableInstance - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

MIDIEnableInstance

MIDIEnableInstance - Syntax

This function enables an instance for sending messages, receiving messages, or both.

```
#include <mididll.h>

MINSTANCE    minstance; /* Instance number. */
ULONG        ulFlag;    /* Flag. */
ULONG        rc;        /* Return code. */

rc = MIDIEnableInstance(minstance, ulFlag);
```

MIDIEnableInstance Parameter - minstance

minstance ([MINSTANCE](#)) - input
The instance to be enabled.

MIDIEnableInstance Parameter - ulFlag

ulFlag ([ULONG](#)) - input
Flag indicating the hardware node should be enabled for sending or receiving messages. Valid values are MIDI_ENABLE_SEND and MIDI_ENABLE_RECEIVE.

MIDIEnableInstance Return Value - rc

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_HARDWARE_FAILED
The enable or open attempt failed due to a hardware problem.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is invalid.

MIDIERR_INVALID_INSTANCE_NUMBER
The instance number (*minstance*) is invalid.

MIDIERR_NOT_ALLOWED
Both send and receive are not permitted for this particular instance.

MIDIERR_RECEIVEONLY
An attempt was made to enable an instance for send (or for send and receive) when the instance supports receive only.

MIDIERR_RESOURCE_NOT_AVAILABLE
The enable or open attempt failed due to a system resource not being available.

MIDIERR_SENDOONLY
An attempt was made to enable an instance for receive (or for send and receive) when the instance supports send only.

MIDIEnableInstance - Parameters

minstance ([MINSTANCE](#)) - input
The instance to be enabled.

ulFlag ([ULONG](#)) - input
Flag indicating the hardware node should be enabled for sending or receiving messages. Valid values are MIDI_ENABLE_SEND and MIDI_ENABLE_RECEIVE.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_HARDWARE_FAILED	The enable or open attempt failed due to a hardware problem.
MIDIERR_INTERNAL_SYSTEM	System problem.
MIDIERR_INVALID_FLAG	The <i>ulFlag</i> value is invalid.
MIDIERR_INVALID_INSTANCE_NUMBER	The instance number (<i>minstance</i>) is invalid.
MIDIERR_NOT_ALLOWED	Both send and receive are not permitted for this particular instance.
MIDIERR_RECEIVEONLY	An attempt was made to enable an instance for send (or for send and receive) when the instance supports receive only.
MIDIERR_RESOURCE_NOT_AVAILABLE	The enable or open attempt failed due to a system resource not being available.
MIDIERR_SENDOONLY	An attempt was made to enable an instance for receive (or for send and receive) when the instance supports send only.

MIDIEnableInstance - Remarks

Instances are enabled separately for send and receive. For example, an application can choose to enable an instance for send only. A reference count is kept of how many times an instance is enabled for send, and a similar count is kept for receive.

If the request is for both send and receive, and the instance only supports one direction, then either MIDIERR_SENDOONLY or MIDIERR_RECEIVEONLY is returned (as appropriate), and the instance is not enabled for either direction. If the instance supports neither send nor receive, MIDIERR_NOT_ALLOWED will be returned. Therefore, the application should call [MIDIQueryInstanceList](#) to determine the capabilities of the instance before attempting to enable it.

MIDIEnableInstance has special meaning for hardware nodes. Hardware nodes are disabled by default and must be explicitly enabled. When a hardware node is enabled for the first time, the Type A driver that it represents is issued an open request. If this request fails then MIDIERR_HARDWARE_FAILED is returned.

All other nodes are enabled by default.

If an application attempts to enable an instance for both directions simultaneously, it is possible for one direction to succeed and for the other to fail. In this case, the instance will be enabled for only one direction and the application will not know which direction that is. This can happen when enabling a hardware node or when the reference count for one direction (but not the other) exceeds an internal limit. In the former case, MIDIERR_HARDWARE_FAILED is returned and in the latter case, MIDIERR_RESOURCE_NOT_AVAILABLE is returned.

Therefore, it is suggested that an application make separate MIDIEnableInstance calls to enable the instance for both directions. If the first call fails, the application can then take appropriate measures.

MIDIEnableInstance - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDIQueryClassList

MIDIQueryClassList - Syntax

This function returns the names of all current classes in *paClassInfo*.

```
#include <mididll.>"

ULONG          ulNumClasses; /* Number of classes. */
PMIDICLASSINFO paClassInfo; /* Pointer to MIDICLASSINFO. */
ULONG          ulFlag;      /* Not used. */
ULONG          rc;          /* Return code. */

rc = MIDIQueryClassList(ulNumClasses, paClassInfo,
                        ulFlag);
```

MIDIQueryClassList Parameter - ulNumClasses

ulNumClasses ([ULONG](#)) - input
Number of classes.

MIDIQueryClassList Parameter - paClassInfo

paClassInfo ([PMIDICLASSINFO](#)) - output
Pointer to an array of [MIDICLASSINFO](#) structures containing the class number and corresponding class name for each class. The number of elements in the array is equal to the *ulNumClasses* value.

MIDIQueryClassList Parameter - ulFlag

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

MIDIQueryClassList Return Value - rc

rc ([ULONG](#)) - returns

Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_CLASS_NUMBER
The *ulNumClasses* value is 0 or exceeds the number of classes.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *paClassInfo* pointer is invalid.

MIDIQueryClassList - Parameters

ulNumClasses ([ULONG](#)) - input
Number of classes.

paClassInfo ([PMIDICLASSINFO](#)) - output
Pointer to an array of [MIDICLASSINFO](#) structures containing the class number and corresponding class name for each class. The number of elements in the array is equal to the *ulNumClasses* value.

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_CLASS_NUMBER
The *ulNumClasses* value is 0 or exceeds the number of classes.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *paClassInfo* pointer is invalid.

MIDIQueryClassList - Remarks

The application should call [MIDIQueryNumClasses](#) to retrieve the number of node classes currently registered and initialize *ulNumClasses* with this value (to determine the size of the *paClassInfo* array).

A Type B driver can register or deregister classes at any time, usually at the prompting of another RTMIDI application. Therefore, an application should be prepared to refresh its list of class names. For simplicity, the application can provide a user-driven method to perform the refresh.

MIDIQueryClassList - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDIQueryInstanceList

MIDIQueryInstanceList - Syntax

This function returns a sequential list of all relevant instance data for all instances in the node network.

```
#include <mididll.h>

ULONG          ulNumInstances; /* Number of instances. */
PMIDIINSTANCEINFO paInstanceInfo; /* Pointer to array of structures. */
ULONG          ulFlag; /* Not used. */
ULONG          rc; /* Return code. */

rc = MIDIQueryInstanceList(ulNumInstances,
                           paInstanceInfo, ulFlag);
```

MIDIQueryInstanceList Parameter - ulNumInstances

ulNumInstances ([ULONG](#)) - input
The number of instances to query.

MIDIQueryInstanceList Parameter - paInstanceInfo

paInstanceInfo ([PMIDIINSTANCEINFO](#)) - output
A pointer to an array of [MIDIINSTANCEINFO](#) structures containing the instance number, class number, instance name, number of links, and instance attributes. The number of elements in the array is equal to *ulNumInstances*.

MIDIQueryInstanceList Parameter - ulFlag

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

MIDIQueryInstanceList Return Value - rc

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *palInstanceInfo* pointer is invalid or the number of instances (*ulNumInstances*) is invalid.

MIDIQueryInstanceList - Parameters

ulNumInstances ([ULONG](#)) - input
The number of instances to query.

palInstanceInfo ([PMIDIINSTANCEINFO](#)) - output
A pointer to an array of [MIDIINSTANCEINFO](#) structures containing the instance number, class number, instance name, number of links, and instance attributes. The number of elements in the array is equal to *ulNumInstances*.

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *palInstanceInfo* pointer is invalid or the number of instances (*ulNumInstances*) is invalid.

MIDIQueryInstanceList - Remarks

The application should call [MIDIQueryNumInstances](#) to retrieve the number of instances and initialize *ulNumInstances* with this value.

The instance names of hardware nodes are defined by device drivers.

The application should provide a user-driven method to refresh the list of instances. The selection of instances can change at any time. Generally, an application would only call `MIDIQueryInstanceList` to obtain the list of hardware nodes; instances of other classes are usually not of interest. Therefore, this function should be called every time the application wants to present a list of available hardware nodes to the user. This ensures that the list is always up to date.

MIDIQueryInstanceList - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDIQueryInstanceName

MIDIQueryInstanceName - Syntax

This function returns the name of an instance.

```
#include <mididll.h>

MINSTANCE    minstance;    /* Instance number. */
PSZ          pszInstanceName; /* Instance name. */
ULONG        ulFlag;       /* Not used. */
ULONG        rc;           /* Return code. */

rc = MIDIQueryInstanceName(minstance, pszInstanceName,
                           ulFlag);
```

MIDIQueryInstanceName Parameter - minstance

minstance ([MINSTANCE](#)) - input
The instance to be queried.

MIDIQueryInstanceName Parameter - pszInstanceName

pszInstanceName ([PSZ](#)) - output
Pointer to a string of length MIDI_NAME_LENGTH containing the instance name.

MIDIQueryInstanceName Parameter - ulFlag

ulFlag (**ULONG**) - input
This parameter is not currently used and must be set to 0.

MIDIQueryInstanceName Return Value - rc

rc (**ULONG**) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_INSTANCE_NUMBER
The instance number (*minstance*) is invalid.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *pszInstanceName* pointer is invalid.

MIDIQueryInstanceName - Parameters

minstance (**MINSTANCE**) - input
The instance to be queried.

pszInstanceName (**PSZ**) - output
Pointer to a string of length MIDI_NAME_LENGTH containing the instance name.

ulFlag (**ULONG**) - input
This parameter is not currently used and must be set to 0.

rc (**ULONG**) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_INSTANCE_NUMBER
The instance number (*minstance*) is invalid.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *pszInstanceName* pointer is invalid.

MIDIQueryInstanceName - Remarks

MIDIQueryInstanceName may also be used to determine if a particular instance number is valid.

MIDIQueryInstanceName - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDIQueryNumClasses

MIDIQueryNumClasses - Syntax

This function returns the number of node classes currently registered.

```
#include <mididll.h>

PULONG    pulNumClasses; /* Number of node classes. */
ULONG     ulFlag;        /* Not used. */
ULONG     rc;            /* Return code. */

rc = MIDIQueryNumClasses(pulNumClasses, ulFlag);
```

MIDIQueryNumClasses Parameter - pulNumClasses

pulNumClasses (**PULONG**) - output
The number of node classes.

MIDIQueryNumClasses Parameter - ulFlag

ulFlag (**ULONG**) - input
This parameter is not currently used and must be set to 0.

MIDIQueryNumClasses Return Value - rc

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *pulNumClasses* pointer is invalid.

MIDIQueryNumClasses - Parameters

pulNumClasses ([PULONG](#)) - output
The number of node classes.

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *pulNumClasses* pointer is invalid.

MIDIQueryNumClasses - Remarks

Once the number of node classes is determined, the application can call [MIDIQueryClassList](#) to obtain a list of class names.

MIDIQueryNumClasses - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

MIDIQueryNumInstances

MIDIQueryNumInstances - Syntax

This function returns the number of instances.

```
#include <mididll.h>

PULONG    pulNumInstances; /* Number of instances. */
ULONG     ulFlag;          /* Not used. */
ULONG     rc;              /* Return code. */

rc = MIDIQueryNumInstances(pulNumInstances,
                           ulFlag);
```

MIDIQueryNumInstances Parameter - pulNumInstances

pulNumInstances (**PULONG**) - output
The number of instances.

MIDIQueryNumInstances Parameter - ulFlag

ulFlag (**ULONG**) - input
This parameter is not currently used and must be set to 0.

MIDIQueryNumInstances Return Value - rc

rc (**ULONG**) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *pulNumInstances* pointer is invalid.

MIDIQueryNumInstances - Parameters

pulNumInstances ([PULONG](#)) - output
The number of instances.

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

[MIDIERR_INTERNAL_SYSTEM](#)
System problem.

[MIDIERR_INVALID_FLAG](#)
The *ulFlag* value is not set to 0.

[MIDIERR_INVALID_PARAMETER](#)
The *pulNumInstances* pointer is invalid.

MIDIQueryNumInstances - Remarks

Once the number of instances is determined, the application can call [MIDIQueryInstanceList](#) to obtain a list of all the current instances.

MIDIQueryNumInstances - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

MIDIQueryVersion

MIDIQueryVersion - Syntax

This function queries the current RTMIDI device driver and DLL version numbers.

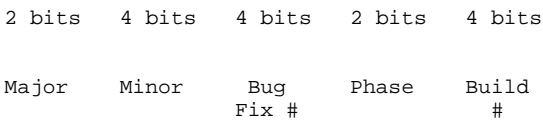
```
#include <mididll.h>

PULONG    pulVersion; /* Version number. */
ULONG     rc;         /* Return code. */

rc = MIDIQueryVersion(pulVersion);
```

MIDIQueryVersion Parameter - pulVersion

pulVersion ([PULONG](#)) - output
The lower 16 bits indicate the device driver version number and the upper 16 bits indicate the DLL version number. The 16 bits are layed out in this format:



Phase is one of the following:

- Development (00)
- Beta (01)
- Alpha (10)
- GA (11)

MIDIQueryVersion Return Value - rc

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

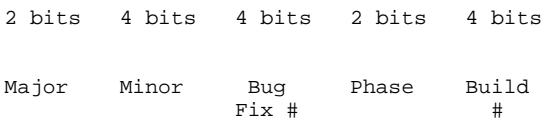
MIDIERR_INVALID_PARAMETER
The *pulVersion* pointer is invalid.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_NO_DRIVER
No device driver.

MIDIQueryVersion - Parameters

pulVersion ([PULONG](#)) - output
The lower 16 bits indicate the device driver version number and the upper 16 bits indicate the DLL version number. The 16 bits are layed out in this format:



Phase is one of the following:

- Development (00)
- Beta (01)
- Alpha (10)
- GA (11)

rc ([ULONG](#)) - returns

Returns 0 if the command is completed successfully.

MIDIERR_INVALID_PARAMETER

The *pulVersion* pointer is invalid.

MIDIERR_INTERNAL_SYSTEM

System problem.

MIDIERR_NO_DRIVER

No device driver.

MIDIQueryVersion - Remarks

The application should call this function first to determine if RTMIDI is up and running.

MIDIQueryVersion - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

MIDIRemoveLink

MIDIRemoveLink - Syntax

This function removes a link from the source instance (*minstanceSource*) to the target instance (*minstanceTarget*).

```
#include <mididll.h>
```

```
MINSTANCE    minstanceSource; /* Source. */
MINSTANCE    minstanceTarget; /* Target. */
ULONG        ulSlotNumber;    /* Slot number. */
ULONG        ulFlag;          /* Not used. */
ULONG        rc;              /* Return code. */
```

```
rc = MIDIRemoveLink(minstanceSource, minstanceTarget,
                    ulSlotNumber, ulFlag);
```

MIDIRemoveLink Parameter - minstanceSource

minstanceSource ([MINSTANCE](#)) - input
The source instance number.

MIDIRemoveLink Parameter - minstanceTarget

minstanceTarget ([MINSTANCE](#)) - input
The target instance number.

MIDIRemoveLink Parameter - ulSlotNumber

ulSlotNumber ([ULONG](#)) - input
The slot number of the source instance for the link to be removed.

MIDIRemoveLink Parameter - ulFlag

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

MIDIRemoveLink Return Value - rc

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INVALID_INSTANCE_NUMBER
Either the source or target instance number is invalid.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *ulSlotNumber* parameter contains an invalid value.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_NOT_ALLOWED

An attempt was made to remove a link which was not created by this process.

MIDIRemoveLink - Parameters

minstanceSource ([MINSTANCE](#)) - input
The source instance number.

minstanceTarget ([MINSTANCE](#)) - input
The target instance number.

uiSlotNumber ([ULONG](#)) - input
The slot number of the source instance for the link to be removed.

uiFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INVALID_INSTANCE_NUMBER
Either the source or target instance number is invalid.

MIDIERR_INVALID_FLAG
The *uiFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *uiSlotNumber* parameter contains an invalid value.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_NOT_ALLOWED
An attempt was made to remove a link which was not created by this process.

MIDIRemoveLink - Remarks

The process removing the link must be the same as the process that created it.

MIDIRemoveLink - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDISendMessages

MIDISendMessages - Syntax

This function sends a block of compound messages.

```
#include <mididll.h>

PMESSAGE    paMessage;    /* Messages. */
ULONG       ulNumMessages; /* Number of messages. */
ULONG       ulFlag;        /* Not used. */
ULONG       rc;            /* Return code. */

rc = MIDISendMessages(paMessage, ulNumMessages,
                      ulFlag);
```

MIDISendMessages Parameter - paMessage

paMessage (**PMESSAGE**) - input
A pointer to an array of compound messages.

MIDISendMessages Parameter - ulNumMessages

ulNumMessages (**ULONG**) - input
The number of compound messages in the array.

MIDISendMessages Parameter - ulFlag

ulFlag (**ULONG**) - input
This parameter is not currently used and must be set to 0.

MIDISendMessages Return Value - rc

rc (**ULONG**) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem or the RTMIDI timer is not started.

MIDIERR_INVALID_INSTANCE_NUMBER
Invalid instance number.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *paMessage* pointer is invalid.

MIDISendMessages - Parameters

paMessage ([PMESSAGE](#)) - input
A pointer to an array of compound messages.

ulNumMessages ([ULONG](#)) - input
The number of compound messages in the array.

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem or the RTMIDI timer is not started.

MIDIERR_INVALID_INSTANCE_NUMBER
Invalid instance number.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *paMessage* pointer is invalid.

MIDISendMessages - Remarks

MIDISendMessages is used to send regular (non-SysEx) MIDI messages only.

The RTMIDI timer *must* be started before calling this function. Call [MIDITimer](#) to start the RTMIDI timer.

The instance number of the application node from which you want to send messages should be set in the *ulSourceInstance* field of the [MESSAGE](#) structures. Since each compound message has its own source instance number, it is possible to mix and match source application nodes within the array of compound messages.

Messages with time stamps indicating the current time or prior will be sent immediately. Messages with time stamps indicating a future time will be placed in a queue.

The time stamp value is absolute. The RTMIDI timer should be used as the basis for the time stamp value. For example, to indicate that a compound message should be processed 10 milliseconds in the future, the value placed in the *ulTime* field of the [MESSAGE](#) structure should be equal to the RTMIDI timer value plus 10.

The application's thread is blocked until all messages are either sent or queued.

If the outbound message queue cannot hold all future messages, the application's thread will be blocked for an indeterminate length of time. While this thread is blocked, the buffer holding the array of messages should not be modified by another thread.

MIDISendMessages - Example Code

The following example illustrates how to send a Note-On MIDI message 10ms into the future.

```
MESSAGE message;      /* The message to send          */
PULONG pulMIDITimer;  /* Initialized from MIDISetup() */
ULONG ulAppInstance;  /* Our application instance      */
ULONG rc;             /* The return code               */

message.ulSourceInstance = ulAppInstance;
message.ulTrack = 0;
message.msg.bytes.bStatus = 0x90; /* Note on, channel 0 */
message.msg.abData[0] = 0x40; /* Note number */
message.msg.abData[1] = 0x7F; /* Maximum volume */
message.ulTime = *pulMIDITimer + 10; /* Play this note 10ms */
/* in the future */

rc = MIDISendMessages(&message,
                     1, /* only one message */
                     0);
```

MIDISendMessages - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

MIDISetup

MIDISetup - Syntax

This function provides a generic setup for an application.

```
#include <mididl1.h>

PMIDISETUP    pMidiSetup; /* Pointer to setup values. */
ULONG         ulFlag;     /* Not used. */
ULONG         rc;         /* Return code. */

rc = MIDISetup(pMidiSetup, ulFlag);
```

MIDISetup Parameter - pMidiSetup

pMidiSetup ([PMIDISETUP](#)) - in/out

A pointer to a [MIDISETUP](#) structure containing the application's MIDI setup values. See [MIDISETUP](#) for detailed descriptions of these values.

MIDISetup Parameter - ulFlag

ulFlag ([ULONG](#)) - input

This parameter is not currently used and must be set to 0.

MIDISetup Return Value - rc

rc ([ULONG](#)) - returns

Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *pMidiSetup* pointer is invalid.

MIDIERR_INVALID_SETUP
Invalid values in [MIDISETUP](#) structure.

MIDISetup - Parameters

pMidiSetup ([PMIDISETUP](#)) - in/out

A pointer to a [MIDISETUP](#) structure containing the application's MIDI setup values. See [MIDISETUP](#) for detailed descriptions of these values.

ulFlag ([ULONG](#)) - input

This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns

Returns 0 if the command is completed successfully.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *pMidiSetup* pointer is invalid.

MIDIERR_INVALID_SETUP
Invalid values in [MIDISETUP](#) structure.

MIDISetup - Remarks

The current implementation of MIDISetup provides the functionality to obtain the maximum length of a real-time SysEx message and a pointer to the RTMIDI timer.

Do not confuse the RTMIDI timer with the timer returned from [TimerGetPointer](#). The pointer returned by MIDISetup is a pointer to the RTMIDI timer, as opposed to the pointer returned from [TimerGetPointer](#), which is a pointer to the high-resolution timer (HRT).

The RTMIDI timer should be used to provide values for the timestamp field (*ulTime* of the [MESSAGE](#) structure) for the [MIDISendMessages](#) function.

This timer can be synchronized to external events, such as SMPTE time clock or OS/2 multimedia, which is why it must be used instead of the high-resolution timer.

MIDISetup - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

MIDITimer

MIDITimer - Syntax

This function provides the ability to start or stop the RTMIDI timer.

```
#include <mididll.h>

ULONG    ulAction; /* Start or stop. */
ULONG    ulFlag;    /* Not used. */
ULONG    rc;        /* Return code. */

rc = MIDITimer(ulAction, ulFlag);
```

MIDITimer Parameter - ulAction

ulAction (ULONG) - input
Indicates whether to start or stop the time. A valid value for *ulAction* is either MIDI_START_TIMER or MIDI_STOP_TIMER.

MIDITimer Parameter - ulFlag

ulFlag (ULONG) - input
This parameter is not currently used and must be set to 0.

MIDITimer Return Value - rc

rc (ULONG) - returns
Returns 0 if the command is completed successfully.

MIDIERR_HARDWARE_FAILED
A hardware specific problem was encountered.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER
The *ulAction* parameter contains an invalid value.

MIDITimer - Parameters

ulAction (ULONG) - input
Indicates whether to start or stop the time. A valid value for *ulAction* is either MIDI_START_TIMER or MIDI_STOP_TIMER.

ulFlag (ULONG) - input
This parameter is not currently used and must be set to 0.

rc (ULONG) - returns
Returns 0 if the command is completed successfully.

MIDIERR_HARDWARE_FAILED
A hardware specific problem was encountered.

MIDIERR_INTERNAL_SYSTEM
System problem.

MIDIERR_INVALID_FLAG
The *ulFlag* value is not set to 0.

MIDIERR_INVALID_PARAMETER

The *ulAction* parameter contains an invalid value.

MIDITimer - Remarks

MIDI_START_TIMER and MIDI_STOP_TIMER are mutually exclusive.

MIDITimer - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Glossary](#)

TimerGetPointer

TimerGetPointer - Syntax

This function obtains the pointer to the high-resolution timer (HRT).

```
#include "mididll.h"

PPULONG    ppulHRTCurrentTime; /* HRT timer. */
ULONG      ulFlag;             /* Not used. */
ULONG      rc;                 /* Return code. */

rc = TimerGetPointer(ppulHRTCurrentTime, ulFlag);
```

TimerGetPointer Parameter - ppulHRTCurrentTime

ppulHRTCurrentTime (PPULONG) - output
Pointer to a pointer to the high-resolution timer.

TimerGetPointer Parameter - ulFlag

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

TimerGetPointer Return Value - rc

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

TIMERERR_INVALID_PARAMETER
Invalid parameter (*ulFlag* not set to 0) or invalid pointer.

TIMERERR_INTERNAL_SYSTEM
System problem.

TimerGetPointer - Parameters

ppulHRTCurrentTime (PPULONG) - output
Pointer to a pointer to the high-resolution timer.

ulFlag ([ULONG](#)) - input
This parameter is not currently used and must be set to 0.

rc ([ULONG](#)) - returns
Returns 0 if the command is completed successfully.

TIMERERR_INVALID_PARAMETER
Invalid parameter (*ulFlag* not set to 0) or invalid pointer.

TIMERERR_INTERNAL_SYSTEM
System problem.

TimerGetPointer - Remarks

The pointer returned by `TimerGetPointer` is a pointer to the high-resolution timer, as opposed to the pointer returned from [MIDISetup](#), which is a pointer to the RTMIDI timer.

The high-resolution timer, unlike the RTMIDI timer, is not synchronized to any external events. In particular, this timer should not be used to provide values for the timestamps required by the [MIDISendMessages](#) function.

TimerGetPointer - Topics

Select an item:
[Syntax](#)
[Parameters](#)

TimerSleep

TimerSleep - Syntax

This function blocks the current thread for the specified number of milliseconds.

```
#include "mididl1.h"

ULONG    ulDuration; /* Duration in ms. */
ULONG    ulFlag;      /* Not used. */
ULONG    rc;          /* Return code. */

rc = TimerSleep(ulDuration, ulFlag);
```

TimerSleep Parameter - ulDuration

ulDuration (**ULONG**) - input
Duration in milliseconds.

TimerSleep Parameter - ulFlag

ulFlag (**ULONG**) - input
This parameter is not currently used and must be set to 0.

TimerSleep Return Value - rc

rc (**ULONG**) - returns
Returns 0 if the command is completed successfully.

TIMERERR_INVALID_PARAMETER
Invalid parameter (*ulFlag* not set to 0).

TIMERERR_INTERNAL_SYSTEM

System problem.

TimerSleep - Parameters

ulDuration (**ULONG**) - input
Duration in milliseconds.

ulFlag (**ULONG**) - input
This parameter is not currently used and must be set to 0.

rc (**ULONG**) - returns
Returns 0 if the command is completed successfully.

TIMERERR_INVALID_PARAMETER
Invalid parameter (*ulFlag* not set to 0).

TIMERERR_INTERNAL_SYSTEM
System problem.

TimerSleep - Remarks

TimerSleep uses the HRT timer, which is not synchronized to any events. This function must be called from a time-critical thread. See DosSetPriority in the *Control Program Programming Guide and Reference* for information on changing a thread's priority.

TimerSleep - Example Code

This example demonstrates how to call a particular function at a periodic rate. MyWorkerFunction() is called every four milliseconds. If the function returns false (zero), the loop is terminated.

```
int MyWorkerFunction(void);

APIRET rc;
ULONG rc2;
ULONG ulSleepDuration = 4;
int fLoop;

/* Change this thread's priority to time-critical */
rc = DosSetPriority(PRTYS_THREAD, PRTYC_TIMECRITICAL, 0, 0);

/* Keep looping until fLoop becomes zero. */
do
{
    rc2 = TimerSleep(ulSleepDuration, 0); /* Sleep */
    fLoop = MyWorkerFunction();          /* Call the worker */
}
while (fLoop);
```

TimerSleep - Topics

Select an item:

- Syntax
- Parameters
- Returns
- Remarks
- Example Code
- Glossary

SPI Functions

The Stream Programming Interface (SPI) contains functions exported by the Sync/Stream Manager to support applications that control real-time data streaming. These functions are used to create data streams, associate data objects, and for data streaming and synchronization. SPI functions also enable applications to query and install stream protocols and to query the current stream time.

The following table lists the SPI functions:

Function	Description
SpiAssociate	Associates a data object with a stream handler.
SpiCreateStream	Creates a stream instance between a source and a target stream handler.
SpiDestroyStream	Removes a stream instance from the system.
SpiDetermineSyncMaster	Determines best master stream to be used in a sync group.
SpiDisableEvent	Disables event detection for a particular event.
SpiDisableSync	Removes a group of streams.
SpiEnableEvent	Enables event detection for a particular event.
SpiEnableSync	Establishes a group of streams to synchronize the streams.
SpiEnumerateHandlers	Returns a list of the stream handler names installed in the SPI.INI file.
SpiEnumerateProtocols	Returns a list of stream protocol keys for the specified stream handler.
SpiGetHandler	Returns the handler ID for the specified stream handler.
SpiGetProtocol	Queries a stream handler for a specified stream protocol.
SpiGetTime	Queries the current stream time.
SpiInstallProtocol	Installs or removes a specified stream protocol for a stream handler.
SpiSeekStream	Seeks to a specified point in the stream object or sets the current stream time.
SpiStartStream	Starts data streaming for a single stream instance or a group of

	streams.
<code>SpiSendMsg</code>	Sends a message to a stream handler.
<code>SpiStopStream</code>	Stops data streaming for a single stream instance or a group of streams.

SpiAssociate

SpiAssociate - Syntax

This function associates a data object with a stream handler.

```
#include <os2.h>

HSTREAM    hstream; /* Stream handle. */
HID        hid;     /* Stream handler ID. */
PACB       pacb;    /* Pointer to control block. */
ULONG      rc;      /* Return codes. */

rc = SpiAssociate(hstream, hid, pacb);
```

SpiAssociate Parameter - hstream

hstream (`HSTREAM`) - input
Stream handle.

SpiAssociate Parameter - hid

hid (`HID`) - input
ID of the source or target stream handler with which to associate.

SpiAssociate Parameter - pacb

pacb ([PACB](#)) - input

Pointer to the parameter packet that describes the object to associate with the handler. Each stream handler defines associate control block types that can be used.

SpiAssociate Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_OBJTYPE

The object type specified is unknown.

ERROR_INVALID_REQUEST

The stream handler does not support the associate function.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_ACCESS_OBJECT

Cannot access the specified data object.

ERROR_STREAM_NOT_STOP

The stream cannot perform the requested function unless the stream is stopped.

ERROR_INVALID_BLOCK

Invalid pointer.

ERROR_INVALID_BUFFER_SIZE

Invalid ACB size.

FAILURE

Stream handler-specific error return code.

SpiAssociate - Parameters

hstream ([HSTREAM](#)) - input

Stream handle.

hid ([HID](#)) - input

ID of the source or target stream handler with which to associate.

pacb ([PACB](#)) - input

Pointer to the parameter packet that describes the object to associate with the handler. Each stream handler defines associate control block types that can be used.

rc ([ULONG](#)) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.	
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_OBJTYPE	The object type specified is unknown.
ERROR_INVALID_REQUEST	The stream handler does not support the associate function.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_ACCESS_OBJECT	Cannot access the specified data object.
ERROR_STREAM_NOT_STOP	The stream cannot perform the requested function unless the stream is stopped.
ERROR_INVALID_BLOCK	Invalid pointer.
ERROR_INVALID_BUFFER_SIZE	Invalid ACB size.
FAILURE	Stream handler-specific error return code.

SpiAssociate - Remarks

Once a stream is created, a data object can be associated with the stream. This data object is what will be streamed to an output device or from an input device. Devices must be specified by using device control blocks (DCBs) in the [SpiCreateStream](#) call. Each stream handler can define a set of associate control blocks that it can understand. The *hid* indicates which stream handler will process this association. This function is valid only on a newly created (not started) stream, a stopped stream (discard or flush stop), or after the end-of-stream notification. This function is not valid after a pause. This function returns an error if the stream has been prerolled, but not started or discard stopped. The stream time is not reset to 0 after a new object is associated. Use [SpiSeekStream](#) to seek to the beginning of the data object. Stream time is no longer valid after an SpiAssociate call. An [SpiSeekStream](#) call must be made by the application media control device to set the correct stream time or to seek the device.

SpiAssociate - Related Functions

- [SpiCreateStream](#)

SpiAssociate - Related Messages

- [SHC_ASSOCIATE](#)

SpiAssociate - Example Code

The following code illustrates how to create a data stream from the file system to the digital audio card.

```
#define          INCL_ERRORS
#include         "os2.h"
#include         "os2me.h"
#include         "mcd.h"
#include         "audio.h"

ULONG          ulRC;                /* Error return code */
HID            hidSource;           /* Source handler ID */
HSTREAM        hStream;            /* Stream handle */
HMMIO          hmmioIn;            /* Handle to MMIO file */
ACB_MMIO       acb;                /* Associate control */
/* block used to */
/* Associate the file to */
/* stream (play) */

SZ             RecordFileName[260] = {"d:\mmos2
\data\mumme.wav"};

.
.
.

/*-----*/
/* Create a data stream from the file system to the digital */
/* connected to the speakers. (See SpiCreateStream.) */
/*-----*/

.
.
.

/*-----*/
/* USE MMIO to access a digital audio object. */
/*-----*/

if ((hmmioIn = mmioOpen(PlayFileName,
                        NULL,
                        MMIO_READWRITE|
                        MMIO_DENYNONE)) == NULL)
    return(ERROR_FILE_NOT_FOUND); /* error! */

/*-----*/
/* Fill in acb with data object information. */
/*-----*/
    acb.ulACBLen = sizeof(ACB_MMIO);
    acb.ulObjType = ACBTYPE_MMIO;
    acb.hmmio = hmmioIn;

/*-----*/
/* Associate the digital audio data object with the source handler. */
/*-----*/
    if (ulRC = SpiAssociate (hStream, hidSource, (PACB)&acb))
        return (ulRC);
```

SpiAssociate - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

SpiCreateStream

SpiCreateStream - Syntax

This function creates a stream instance between a source and a target stream handler.

```
#include <os2.h>

HID          hidSrc;      /* Source stream handler ID. */
HID          hidTgt;      /* Target stream handler ID. */
PSPCBKEY     pspcbkey;    /* Data type (subtype) to be streamed. */
PDCB         pdcbsrc;     /* Pointer to device control block (source). */
PDCB         pdcbtgt;     /* Pointer to device control block (target). */
PIMPL_EVCB   pevcb;       /* Pointer to EVCB for implicit events. */
PEVFN        EventEntry;  /* Entry point. */
HSTREAM      hstreamBuf;  /* Stream handle. */
PHSTREAM     phstream;    /* Stream handler address. */
PHEVENT      phevent;     /* Implicit event handle. */
ULONG        rc;          /* Return codes. */

rc = SpiCreateStream(hidSrc, hidTgt, pspcbkey,
                    pdcbsrc, pdcbtgt, pevcb, EventEntry,
                    hstreamBuf, phstream, phevent);
```

SpiCreateStream Parameter - hidSrc

hidSrc ([HID](#)) - input
Source stream handler ID.

SpiCreateStream Parameter - hidTgt

hidTgt ([HID](#)) - input
Target stream handler ID.

SpiCreateStream Parameter - pspcbkey

pspcbkey ([PSPCBKEY](#)) - input
Data type (subtype) to be streamed.

SpiCreateStream Parameter - pdcbSrc

pdcbSrc ([PDCB](#)) - input
Pointer to the device control block that contains device-specific instance information about the source device. This field must be NULL for stream handlers that do not support a DCB.

SpiCreateStream Parameter - pdcbTgt

pdcbTgt ([PDCB](#)) - input
Pointer to device control block that contains device-specific instance information about the target device. This field must be NULL for stream handlers that do not support a DCB.

SpiCreateStream Parameter - pevcb

pevcb ([PIMPL_EVCB](#)) - input
Structure is filled in during event notification. The structure must be allocated statically and can not be a local variable.

SpiCreateStream Parameter - EventEntry

EventEntry ([PEVFN](#)) - input
Entry point of a routine to handle stream events. This routine is called whenever an event is reported to or generated by the Sync/Stream Manager. All events reported are serialized; therefore, it is advantageous to keep the event routine as small as possible. Some SPI functions can be called from within an event routine. [SpiDestroyStream](#) should not be issued from within the event routine.

SpiCreateStream Parameter - hstreamBuf

hstreamBuf ([HSTREAM](#)) - input
If the stream protocol for this stream instance specifies that the stream is a split stream, then this field indicates which stream instance has allocated buffers that this stream will use. In this case, the Sync/Stream Manager does not allocate a new set of stream buffers for this stream. If this stream instance is not a split stream, the Sync/Stream Manager ignores this field, but it will be passed to the source and target stream handlers.

SpiCreateStream Parameter - phstream

phstream ([PHSTREAM](#)) - output
Address for handle of newly created stream.

SpiCreateStream Parameter - phevent

phevent ([PHEVENT](#)) - output
Implicit event handle.

SpiCreateStream Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_BLOCK	Invalid pointer.
ERROR_INVALID_BUFFER_SIZE	Invalid DCB size.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_PARAMETER	Invalid parameter.
ERROR_INVALID_PROTOCOL	An error occurred while negotiating the stream protocol for this stream instance.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_SPCBKEY	Invalid <i>pspcbkey</i> . The stream type is unknown or the stream handlers specified cannot stream this type.
ERROR_DEVICE_NOT_FOUND	Invalid device-driver name.
ERROR_ALLOC_HEAP	An error occurred while allocating the resources needed to set up this stream. The heap can be adjusted to a bigger size on the DEVICE= statement for the Sync/Stream Manager device driver. See <i>Performance Tuning the Sync/Stream Manager</i> in the <i>OS/2 Multimedia Subsystem Programming Guide</i> .
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate the minimum number of buffers to sustain this stream instance.

ERROR_TOO_MANY_STREAMS

Too many streams have been created. The number of streams can be adjusted to a bigger size on the DEVICE statement for the Sync/Stream Manager device driver. See *Performance Tuning the Sync/Stream Manager* in the *OS/2 Multimedia Subsystem Programming Guide*.

FAILURE

Stream handler-specific error return code.

SpiCreateStream - Parameters

hidSrc (HID) - input

Source stream handler ID.

hidTgt (HID) - input

Target stream handler ID.

pspcbkey (PSPCBKEY) - input

Data type (subtype) to be streamed.

pdcBsrc (PDCB) - input

Pointer to the device control block that contains device-specific instance information about the source device. This field must be NULL for stream handlers that do not support a DCB.

pdcBTgt (PDCB) - input

Pointer to device control block that contains device-specific instance information about the target device. This field must be NULL for stream handlers that do not support a DCB.

pevcB (PIMPL_EVCB) - input

Structure is filled in during event notification. The structure must be allocated statically and can not be a local variable.

EventEntry (PEVFN) - input

Entry point of a routine to handle stream events. This routine is called whenever an event is reported to or generated by the Sync/Stream Manager. All events reported are serialized; therefore, it is advantageous to keep the event routine as small as possible. Some SPI functions can be called from within an event routine. [SpiDestroyStream](#) should not be issued from within the event routine.

hstreamBuf (HSTREAM) - input

If the stream protocol for this stream instance specifies that the stream is a split stream, then this field indicates which stream instance has allocated buffers that this stream will use. In this case, the Sync/Stream Manager does not allocate a new set of stream buffers for this stream. If this stream instance is not a split stream, the Sync/Stream Manager ignores this field, but it will be passed to the source and target stream handlers.

phstream (PHSTREAM) - output

Address for handle of newly created stream.

phevent (PHEVENT) - output

Implicit event handle.

rc (ULONG) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_BLOCK

Invalid pointer.

ERROR_INVALID_BUFFER_SIZE

Invalid DCB size.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_PARAMETER	Invalid parameter.
ERROR_INVALID_PROTOCOL	An error occurred while negotiating the stream protocol for this stream instance.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_SPCBKEY	Invalid <i>pspcbkey</i> . The stream type is unknown or the stream handlers specified cannot stream this type.
ERROR_DEVICE_NOT_FOUND	Invalid device-driver name.
ERROR_ALLOC_HEAP	An error occurred while allocating the resources needed to set up this stream. The heap can be adjusted to a bigger size on the DEVICE= statement for the Sync/Stream Manager device driver. See <i>Performance Tuning the Sync/Stream Manager</i> in the <i>OS/2 Multimedia Subsystem Programming Guide</i> .
ERROR_NOT_ENOUGH_MEMORY	There is not enough memory to allocate the minimum number of buffers to sustain this stream instance.
ERROR_TOO_MANY_STREAMS	Too many streams have been created. The number of streams can be adjusted to a bigger size on the DEVICE statement for the Sync/Stream Manager device driver. See <i>Performance Tuning the Sync/Stream Manager</i> in the <i>OS/2 Multimedia Subsystem Programming Guide</i> .
FAILURE	Stream handler-specific error return code.

SpiCreateStream - Remarks

Resources for the stream are allocated at this time. Split streams share one set of stream buffers. This type of stream is used for interleaved data types. Once a stream is created of an interleaved data type (specified in the SPCB), another stream can use the stream handle of the first stream in the *hstreamBuf* parameter to indicate that it needs to share the set of buffers of the first stream. Each stream created must have a unique implicit event control block (EVCB). For NULL streams, both the *hidSrc* and the *hidTgt* must be the NULL stream handler ID (HID). Source and target devices can be specified using the DCBs for the respective stream handlers. Each stream handler can define a set of device control blocks that can be used at stream-creation time.

SpiCreateStream - Related Functions

- [SpiGetHandler](#)
- [SpiDestroyStream](#)
- [SpiStartStream](#)
- [SpiStopStream](#)
- [SpiSeekStream](#)
- [SpiEnableEvent](#)
- [SpiGetTime](#)
- [SpiEnumerateProtocols](#)
- [SpiGetProtocol](#)
- [SpiInstallProtocol](#)

SpiCreateStream - Related Messages

- [SHC_CREATE](#)
- [SHC_NEGOTIATE_RESULT](#)

SpiCreateStream - Example Code

The following code illustrates how to create a stream handler between a source and target stream handler.

```
#include "os2.h"
#include "os2me.h"
#include "string.h"

ULONG      ulRC;          /* Error return code */
HID        hidSource,     /* Source handler ID */
           hidTarget,     /* Target handler ID */
           hidunused;     /* Dummy handler ID */

SPCBKEY    spcbkey;       /* Data type to stream */
DCB_AUDIOSH dcb;          /* Audio device driver information */
PDCB       pdcbTgt;       /* Pointer to DCB */
IMPL_EVCB  evcb;          /* Event control block for
                           /* implicit events */

HSTREAM     hStream;      /* Stream handle */
HEVENT      hEvent;       /* Event handle for implicit */
                           /* events */

.
.
.

/*-----*/
/* Initialize the audio card and get the          */
/* dcb.hSysFileNum for "AUDIO01$".                */
/*-----*/

.
.
.

/*-----*/
/* Get the stream handler ID for the file stream handler.*/
/*-----*/
if (ulRC = SpiGetHandler("FSSH",&hidSource, &hidunused))
    return(ulRC);      /* error! */

/*-----*/
/* Get the stream handler ID for the digital audio */
/* stream handler.                                */
/*-----*/
if (ulRC = SpiGetHandler("AUDIOSH",&hidunused, &hidTarget))
    return(ulRC);      /* error! */

/*-----*/
/* Create a data stream from the file system to    */
/* the digital audio card connected to the speakers. */
/*-----*/
spcbkey.ulDataType = DATATYPE_WAVEFORM;
spcbkey.ulDataSubType = WAVE_FORMAT_4S16;
spcbkey.ulIntKey = 0;
strcpy(dcb.szDevName,"AUDIO01$");
pdcbTgt = (PDCB)&dcb;

if (ulRC = SpiCreateStream (hidSource,
                           hidTarget,
                           &spcbkey,
                           NULL,
                           pdcbTgt,
                           &evcb,
                           (PEVFN) EventsRtn,
                           0,
                           &hStream,
                           &hEvent))
    return(ulRC);      /* error! */
```

```

/*-----*/
/*  Event routine                                */
/*-----*/

ULONG APIENTRY EventsRtn(PEVCB pevcb)
{
    .
    .
    Process events.
    .
    .

    return(0);
}

```

SpiCreateStream - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

SpiDestroyStream

SpiDestroyStream - Syntax

This function removes a stream from the system.

```
#include <os2.h>
```

```

HSTREAM    hstream; /* Stream handle. */
ULONG      rc;       /* Return codes. */

```

```
rc = SpiDestroyStream(hstream);
```

SpiDestroyStream Parameter - hstream

hstream ([HSTREAM](#)) - input

Identifies the stream to remove from the system.

SpiDestroyStream Return Value - rc

rc ([ULONG](#)) - returns

Return code indicating success or the type of failure:

NO_ERROR

Success.

ERROR_DESTROY_STREAM

Tried to destroy a split stream that owns buffers while streams that use these buffers still exist.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

FAILURE

Stream handler-specific error return code.

SpiDestroyStream - Parameters

hstream ([HSTREAM](#)) - input

Identifies the stream to remove from the system.

rc ([ULONG](#)) - returns

Return code indicating success or the type of failure:

NO_ERROR

Success.

ERROR_DESTROY_STREAM

Tried to destroy a split stream that owns buffers while streams that use these buffers still exist.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

FAILURE

Stream handler-specific error return code.

SpiDestroyStream - Remarks

After this function is issued for a stream, the stream handle for that stream can no longer be used. All resources for that stream are returned to the system. Removing the master stream of a sync group disables the sync group. Removing a slave stream of a sync group removes only the slave stream from the group, and does not affect the other streams in the group. Split streams require that the stream that owns the buffers be destroyed last.

SpiDestroyStream - Related Functions

- [SpiCreateStream](#)

SpiDestroyStream - Related Messages

- [SHC_DESTROY](#)
- [SHC_DISABLE_SYNC](#)

SpiDestroyStream - Example Code

The following code illustrates how to remove a stream from the system.

```
#include      "os2.h"
#include      "os2me.h"

ULONG        ulRC;                /* Error return code */
HSTREAM      hStream;             /* Stream handle */

        .
        .
        .
/*-----*/
/*  Create a data stream (hStream) and process the data.*/
/*-----*/
        .
        .
        .
/*-----*/
/*  Finished with stream - Now destroy the stream.*/
/*-----*/
if (ulRC = SpiDestroyStream(hStream))
    return (ulRC);    /* error! */
```

SpiDestroyStream - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiDetermineSyncMaster

SpiDetermineSyncMaster - Syntax

This function selects the best master stream to be used in a synchronized group. It allows a super group to be created that may contain other synchronized groups.

```
#include <os2.h>

PHSTREAM    phstreamMaster; /* Handle address. */
PMASTER    paMasterList;  /* Stream handle array address. */
ULONG       ulNumMasters;  /* Number of streams in master list. */
ULONG       rc;            /* Return code. */

rc = SpiDetermineSyncMaster(phstreamMaster,
                           paMasterList, ulNumMasters);
```

SpiDetermineSyncMaster Parameter - phstreamMaster

phstreamMaster (**PHSTREAM**) - output
Return address for the stream handle of the best master stream.

SpiDetermineSyncMaster Parameter - paMasterList

paMasterList (**PMASTER**) - input
Array address of stream handles that select master stream of a synchronized group.

SpiDetermineSyncMaster Parameter - ulNumMasters

ulNumMasters (**ULONG**) - input
Number of streams in *paMasterList*.

SpiDetermineSyncMaster Return Value - rc

rc (**ULONG**) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_NUMMASTERS	Invalid number of streams in <i>uiNumMasters</i> .
ERROR_MASTER_CONFLICT	Stream master conflict, two or more streams have master capabilities only.
ERROR_NO_SYNC	Stream cannot be in a synchronization group.
ERROR_NO_MASTER	No stream can be a master stream.

SpiDetermineSyncMaster - Parameters

phstreamMaster ([PHSTREAM](#)) - output
Return address for the stream handle of the best master stream.

paMasterList ([PMASTER](#)) - input
Array address of stream handles that select master stream of a synchronized group.

uiNumMasters ([ULONG](#)) - input
Number of streams in *paMasterList*.

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_NUMMASTERS	Invalid number of streams in <i>uiNumMasters</i> .
ERROR_MASTER_CONFLICT	Stream master conflict, two or more streams have master capabilities only.
ERROR_NO_SYNC	Stream cannot be in a synchronization group.
ERROR_NO_MASTER	No stream can be a master stream.

SpiDetermineSyncMaster - Remarks

The stream determined to be the best stream suited to be the master of a synchronization group is calculated by using the lowest sync pulse granularity (*mmtimeSync* in [SPCB](#)) used to generate the sync pulses to the slave streams.

SpiDetermineSyncMaster - Related Functions

- [SpiEnableSync](#)
- [SpiDisableSync](#)

SpiDetermineSyncMaster - Related Messages

- [SHC_ENABLE_SYNC](#)
- [SHC_DISABLE_SYNC](#)

SpiDetermineSyncMaster - Example Code

The following code illustrates creating a super group that may contain other synchronization groups.

```
#include "os2.h"
#include "os2me.h"

ULONG      ulRc;          /*error return code */
HSTREAM    hstream1;      /*stream 1 handle */
HSTREAM    hstream2;      /*stream 2 handle */

MASTER     master[2] /*Sync Master Structure */
HSTREAM    hstreamMaster;

/*****
/* Create stream for hstream1 and hstream2. */
/* (See SpiCreateStream) */
*****/

.
.
.

/*****
/* Determine which stream should be used as the master stream */
/* of a synchronized group. */
*****/

master[0].hstreamMaster = HSTREAM1;
master[1].hstreamMaster = HSTREAM2;

if (ulRc = SpiDetermineSyncMaster (&hstreamMaster,
                                   &master [0],
                                   2))

    return(ulRc);          /* error */

/*****
/* Use the handle received in hstreamMaster as the master */
/* stream handle. See SpiEnableSync. */
*****/
```

SpiDetermineSyncMaster - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

SpiDisableEvent

SpiDisableEvent - Syntax

This function disables event notification for a particular event.

```
#include <os2.h>

HEVENT    hevent;    /* Event. */
ULONG     rc;        /* Return codes indicating success or the type of failure: */

rc = SpiDisableEvent(hevent);
```

SpiDisableEvent Parameter - hevent

hevent ([HEVENT](#)) - input
Identifies the event to disable.

SpiDisableEvent Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_EVENT	Invalid event handle

ERROR_INVALID_REQUEST
Tried to disable an implicit event.

FAILURE
Stream handler specific error return code.

SpiDisableEvent - Parameters

hevent ([HEVENT](#)) - input
Identifies the event to disable.

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR
Success.

ERROR_INVALID_FUNCTION
Invalid function requested.

ERROR_INVALID_EVENT
Invalid event handle

ERROR_INVALID_REQUEST
Tried to disable an implicit event.

FAILURE
Stream handler specific error return code.

SpiDisableEvent - Remarks

After this function is issued, the event handle for the event can no longer be used. The event is removed from the system for the particular stream instance. When any given event is reported, it is not removed automatically from the system; it must be removed explicitly using this function. Once an event is removed from the system, the event no longer is detected or reported to the application or media control driver.

SpiDisableEvent - Related Functions

- [SpiEnableEvent](#)
-

SpiDisableEvent - Related Messages

- [SHC_DISABLE_EVENT](#)
-

SpiDisableEvent - Example Code

The following code illustrates how to disable event notification for a particular event.

```
#include      "os2.h"
#include      "os2me.h"

ULONG        ulRC;                                /* Error return code */
HEVENT       hevent;                              /* Event handle */

.
.
.

/*----- */
/*  Create a data stream with an event routine, and enable an event */
/*  to get the event handle (hevent). (See SpiCreateStream and */
/*  SpiEnableEvent.) */
/*----- */

.
.

/*----- */
/*  Do other processing. */
/*----- */

.
.

/*----- */
/*  Now disable the event. */
/*----- */
if (ulRC = SpiDisableEvent (heventTime))
    return (ulRC);    /* error! */
```

SpiDisableEvent - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiDisableSync

SpiDisableSync - Syntax

This function removes a grouping of streams.

```
#include <os2.h>

HSTREAM     hstreamMaster; /* Stream grouping. */
```

```
ULONG      rc;                /* Return codes indicating success or the type of failure: */  
  
rc = SpiDisableSync(hstreamMaster);
```

SpiDisableSync Parameter - hstreamMaster

hstreamMaster ([HSTREAM](#)) - input
Identifies the stream grouping to remove. The group is identified by the master stream in the group.

SpiDisableSync Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_STREAM_NOTMASTER	The stream handle passed was not a synchronization master stream.
ERROR_STREAM_PREROLLING	The sync group cannot be removed while streams in the group are being prerolled.
FAILURE	Stream handler-specific error return code.

SpiDisableSync - Parameters

hstreamMaster ([HSTREAM](#)) - input
Identifies the stream grouping to remove. The group is identified by the master stream in the group.

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_STREAM_NOTMASTER	

The stream handle passed was not a synchronization master stream.

ERROR_STREAM_PREROLLING

The sync group cannot be removed while streams in the group are being prerolled.

FAILURE

Stream handler-specific error return code.

SpiDisableSync - Remarks

Stops stream synchronization for a group of streams. A previously designated stream master is disabled, and all synchronization under that master is deactivated. The streams that were grouped previously can no longer be used as a group.

SpiDisableSync - Related Functions

- [SpiEnableSync](#)
- [SpiDestroyStream](#)

SpiDisableSync - Related Messages

- [SHC_DISABLE_SYNC](#)

SpiDisableSync - Example Code

The following code illustrates how to remove a grouping of streams.

```
#include "os2.h"
#include "os2me.h"

ULONG      ulRC;                /* Error return code */
HSTREAM    hStream1;            /* Stream 1 handle */
.
.
.

/*-----*/
/* Create digital audio stream (hStream1). Create a second digital */
/* audio stream (hStream2). Set up the sync group between the two */
/* streams (See SpiCreateStream, SpiAssociate, and SpiStartStream.) */
/*-----*/
.
.

/*-----*/
/* Do other processing. */
/*-----*/
.
.

/*-----*/
/* Now disable synchronization between the two streams. */
/*-----*/
```

```
/*-----*/
if (ulRC = SpiDisableSync (hStream1))
    return (ulRC);    /* error! */
```

SpiDisableSync - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

SpiEnableEvent

SpiEnableEvent - Syntax

This function enables event notification for a particular event.

```
#include <os2.h>

PEVCB      pevcb;    /* Event type. */
PHEVENT    phevent;  /* Address. */
ULONG      rc;        /* Return codes indicating success or the type of failure: */

rc = SpiEnableEvent(pevcb, phevent);
```

SpiEnableEvent Parameter - pevcb

pevcb ([PEVCB](#)) - input

Identifies the event-specific information (that is, the event type). The EVCB is a generic version of the event control block. Each event can have different field definitions in the EVCB.

SpiEnableEvent Parameter - phevent

phevent (**PHEVENT**) - input
Identifies the address to return the newly created event handle.

SpiEnableEvent Return Value - rc

rc (**ULONG**) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_EVENT	Specified event type is unrecognized.
ERROR_INVALID_EVCB	Event information is invalid for the specified event type and stream.
ERROR_INVALID_BLOCK	Invalid pointer.
ERROR_TOO_MANY_EVENTS	An error occurred while allocating the resources needed to enable this event.
FAILURE	Stream handler-specific error return code.

SpiEnableEvent - Parameters

pevcb (**PEVCB**) - input
Identifies the event-specific information (that is, the event type). The EVCB is a generic version of the event control block. Each event can have different field definitions in the EVCB.

phevent (**PHEVENT**) - input
Identifies the address to return the newly created event handle.

rc (**ULONG**) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_EVENT	Specified event type is unrecognized.
ERROR_INVALID_EVCB	Event information is invalid for the specified event type and stream.

ERROR_INVALID_BLOCK	Invalid pointer.
ERROR_TOO_MANY_EVENTS	An error occurred while allocating the resources needed to enable this event.
FAILURE	Stream handler-specific error return code.

SpiEnableEvent - Remarks

The *EventEntry* routine specified in [SpiCreateStream](#) is called when the event is detected. Multiple events can be defined for each stream instance. This event will be enabled for the stream instance specified in the *hstream* field of [EVCB](#). The Sync/Stream Manager supplies several system-defined events, but a stream handler can define new events. Event types must fall within the numeric range of 0x80000000 to 0xFFFFFFFF. All other event values are reserved for the system. Following is a list of system-defined events that can be enabled:

```
EVENT_SYNCOVERRUN
EVENT_CUE_TIME
EVENT_CUE_TIME_PAUSE
EVENT_CUE_DATA
EVENT_DATAOVERRUN
EVENT_DATAUNDERRUN
```

Each stream handler also can define values for the *ulFlags* field in the [EVCB](#).

SpiEnableEvent - Related Functions

- [SpiDisableEvent](#)
- [SpiDestroyStream](#)

SpiEnableEvent - Related Messages

- [SHC_ENABLE_EVENT](#)

SpiEnableEvent - Example Code

The following code illustrates how to enable event notification for a particular event.

```
#include      "os2.h"
#include      "os2me.h"

ULONG        ulRC;                /* Error return code          */
HID          hidTarget;           /* Source handler ID          */
HSTREAM      hStream;            /* Stream handle              */
TIME_EVCB    TIMEEVCB = [0,0,0,0,0,0,0,0]; /* Cue point event          */
HEVENT       hevent;             /* Event handle               */
```

```

        .
        .
        .
/*-----*/
/* Create a data stream (hStream). (See SpiCreateStream.) */
/*-----*/
        .
        .
        .
/*-----*/
/* Set an event. */
/*-----*/
    TIMEEVCB.ulType = EVENT_CUE_TIME;
    TIMEEVCB.ulFlags = EVENT_SINGLE;          /* Report it once. */
    TIMEEVCB.hstream = hStream;
    TIMEEVCB.hid = hidTarget;                 /* Let target detect event. */
    TIMEEVCB.mmtimeStream = 60000;           /* 20 seconds */

    if (ulRC = SpiEnableEvent ((PEVCB) &TIMEEVCB, &hevent))
        return (ulRC);    /* error! */

```

SpiEnableEvent - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiEnableSync

SpiEnableSync - Syntax

This function establishes a group of streams to synchronize.

```

#include <os2.h>

HSTREAM    hstreamMaster; /* Stream handle. */
PSLAVE     paslaveList;  /* Pointer to an array of slave entries. */
ULONG      ulNumSlaves;   /* Number of slaves. */
MMTIME     mmtimeSync;    /* MMTIME value. */
ULONG      rc;            /* Return codes. */

rc = SpiEnableSync(hstreamMaster, paslaveList,
    ulNumSlaves, mmtimeSync);

```

SpiEnableSync Parameter - hstreamMaster

hstreamMaster ([HSTREAM](#)) - input

Handle to stream that will be the master in the stream group.

SpiEnableSync Parameter - paslaveList

paslaveList ([PSLAVE](#)) - input

Each slave entry provides information about one stream that will be a slave stream in the group. This information includes the stream handle, as well as the *mmtimeStart* field that is used to specify the difference in stream time between the master stream and slave streams. This value is used in the sync tolerance calculations and affects the starting of the slave streams.

SpiEnableSync Parameter - ulNumSlaves

ulNumSlaves ([ULONG](#)) - input

Number of slaves in the *paslaveList* array.

SpiEnableSync Parameter - mmtimeSync

mmtimeSync ([MMTIME](#)) - input

Value indicating the desired sync pulse generation interval. A sync pulse is generated by the master stream at each *mmtimeSync* interval relative to MMTIME 0. MMTIME is always specified in units of 1/3000th of a second. If this parameter is NULL, the sync pulse interval in the negotiated stream protocol control block ([SPCB](#)) for the master stream is used. The SPCB for a data type and stream handler can be queried using the [SpiGetProtocol](#) function.

SpiEnableSync Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_NUMSLAVES	Invalid number of slaves specified (probably 0).
ERROR_INVALID_MMTIME	Invalid MMTIME specified, or the wrong type for the stream master.
ERROR_MASTER_USED	The stream master already specified is a stream master for another sync group. Use SpiDisableSync to remove the other sync group, then try this call again.
ERROR_STREAM_USED	One of the streams specified already is part of another sync group.
ERROR_INVALID_BLOCK	Invalid pointer.
FAILURE	Stream handler-specific error return code.

SpiEnableSync - Parameters

hstreamMaster ([HSTREAM](#)) - input

Handle to stream that will be the master in the stream group.

paslaveList ([PSLAVE](#)) - input

Each slave entry provides information about one stream that will be a slave stream in the group. This information includes the stream handle, as well as the *mmtimeStart* field that is used to specify the difference in stream time between the master stream and slave streams. This value is used in the sync tolerance calculations and affects the starting of the slave streams.

uiNumSlaves ([ULONG](#)) - input

Number of slaves in the *paslaveList* array.

mmtimeSync ([MMTIME](#)) - input

Value indicating the desired sync pulse generation interval. A sync pulse is generated by the master stream at each *mmtimeSync* interval relative to MMTIME 0. MMTIME is always specified in units of 1/3000th of a second. If this parameter is NULL, the sync pulse interval in the negotiated stream protocol control block ([SPCB](#)) for the master stream is used. The SPCB for a data type and stream handler can be queried using the [SpiGetProtocol](#) function.

rc ([ULONG](#)) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_NUMSLAVES

Invalid number of slaves specified (probably 0).

ERROR_INVALID_MMTIME

Invalid MMTIME specified, or the wrong type for the stream master.

ERROR_MASTER_USED

The stream master already specified is a stream master for another sync group. Use [SpiDisableSync](#) to remove the other sync group, then try this call again.

ERROR_STREAM_USED

One of the streams specified already is part of another sync group.

ERROR_INVALID_BLOCK

Invalid pointer.

FAILURE

Stream handler-specific error return code.

SpiEnableSync - Remarks

Stream synchronization is handled automatically by the stream protocol, depending on whether the given stream is the synchronization master or slave. Once a synchronization relationship is established, synchronization is maintained according to real-time events detected at the master and passed to the slaves. The application does not have additional stream events to process because of synchronization. Only one stream can be a master within a sync group.

SpiEnableSync - Related Functions

- [SpiDisableSync](#)
- [SpiStartStream](#)
- [SpiStopStream](#)
- [SpiSeekStream](#)
- [SpiDestroyStream](#)

SpiEnableSync - Related Messages

- [SHC_ENABLE_SYNC](#)

SpiEnableSync - Example Code

The following code illustrates how to establish a group of streams to synchronize.

```
#include      "os2.h"
#include      "os2me.h"

ULONG        ulRC;                /* Error return code */
HSTREAM      hStream1,            /* Stream 1 handle */
             hStream2;            /* Stream 2 handle */
SLAVE        slave[1];            /* Sync slave structure */
            .
            .
/*-----*/
/* Create digital audio stream (hStream1). Create second */
/* digital audio stream (hStream2). (See SpiCreateStream, */
/* SpiAssociate, and SpiStartStream.) */
/*-----*/
            .
            .
/*-----*/
/* Set up synchronization between the two streams. */
/*-----*/
```

```

slave[0].hstreamSlave = hStream2; /* 2nd stream is slave. */
slave[0].mmtimeStart = 0;        /* Time slave will start.*/

if (ulRC = SpiEnableSync (hStream1,          /* Audio stream is master. */
                        &slave[1],
                        1,                    /* Number of slaves */
                        0 ))                /* Use SPCB sync granularity.*/
    return (ulRC);    /* error! */

```

SpiEnableSync - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiEnumerateHandlers

SpiEnumerateHandlers - Syntax

This function returns a list of the stream handler names installed in the SPI.INI file.

```

#include <os2.h>

PHAND    pahand;    /* Buffer pointer. */
PULONG   pulNumHand; /* Handler pointer. */
ULONG    rc;        /* Return codes indicating success or the type of failure: */

rc = SpiEnumerateHandlers(pahand, pulNumHand);

```

SpiEnumerateHandlers Parameter - pahand

pahand ([PHAND](#)) - in/out

Pointer to the buffer to be filled in with stream-handler names and class names.

SpiEnumerateHandlers Parameter - pulNumHand

pulNumHand (**PULONG**) - in/out

Pointer to the number of handlers that can be returned in the *pahand* buffer. If there are more handlers than will fit, an error is returned.

SpiEnumerateHandlers Return Value - rc

rc (**ULONG**) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_BLOCK

Invalid pointer.

ERROR_INVALID_BUFFER_SIZE

Buffer passed is too small to hold all handler names or classes. The number returned in *pulNumHand* is the number of handlers that exist.

ERROR_READING_INI

An error occurred reading the SPI.INI file.

SpiEnumerateHandlers - Parameters

pahand (**PHAND**) - in/out

Pointer to the buffer to be filled in with stream-handler names and class names.

pulNumHand (**PULONG**) - in/out

Pointer to the number of handlers that can be returned in the *pahand* buffer. If there are more handlers than will fit, an error is returned.

rc (**ULONG**) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_BLOCK

Invalid pointer.

ERROR_INVALID_BUFFER_SIZE

Buffer passed is too small to hold all handler names or classes. The number returned in *pulNumHand* is the number of handlers that exist.

ERROR_READING_INI

An error occurred reading the SPI.INI file.

SpiEnumerateHandlers - Remarks

The list of the stream handler names installed in the SPI.INI file includes any registered device driver stream handlers and any available DLL stream handlers accessible by the Sync/Stream Manager. The SPI.INI file is created at installation time. Refer to the *OS/2 Multimedia Subsystem Programming Guide* for more information on installing stream handlers or stream protocols in the SPI.INI file.

SpiEnumerateHandlers - Related Functions

- [SpiGetHandler](#)
 - [SpiEnumerateProtocols](#)
 - [SpiGetProtocol](#)
 - [SpiInstallProtocol](#)
-

SpiEnumerateHandlers - Example Code

The following code illustrates how to return a list of the stream handler names installed in the SPI.INI file.

```
#include      "os2.h"
#include      "os2me.h"

ULONG        ulRC;                /* Error return code */
HAND         ahand[5];            /* Enumerate handler info. */
ULONG        ulNumHand = 5;       /* Number of stream handlers */

/*-----*/
/*  Get list of all stream handlers in system.  */
/*-----*/
if (ulRC = SpiEnumerateHandlers( (PHAND)&ahand, &ulNumHand))
    return(ulRC);                /* error! */
```

SpiEnumerateHandlers - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

SpiEnumerateProtocols

SpiEnumerateProtocols - Syntax

This function returns a list of stream protocol keys for the specified stream handler.

```
#include <os2.h>

HID      hid;           /* Stream handler ID. */
PVOID     paSPCBKeys;   /* SPCB keys address. */
PULONG    pulNumSPCBKeys; /* Number of entries. */
ULONG     rc;           /* Return codes. Return codes indicating success or the type of failure: */

rc = SpiEnumerateProtocols(hid, paSPCBKeys,
                           pulNumSPCBKeys);
```

SpiEnumerateProtocols Parameter - hid

hid ([HID](#)) - input
ID (returned by [SpiGetHandler](#)) whose protocols are to be queried.

SpiEnumerateProtocols Parameter - paSPCBKeys

paSPCBKeys ([PVOID](#)) - in/out
Address of an array of SPCB keys to be filled in that identify the protocols supported by the specified stream handler.

SpiEnumerateProtocols Parameter - pulNumSPCBKeys

pulNumSPCBKeys ([PULONG](#)) - in/out
Pointer to the number of entries on input. Upon return, contains the number actually returned. If more protocols are available than there are entries in the array, this function returns `ERROR_INVALID_BUFFER_SIZE`, and this parameter contains the number of array entries needed.

SpiEnumerateProtocols Return Value - rc

rc ([ULONG](#)) - returns
Return codes. Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_BUFFER_SIZE	Array passed is too small to hold all the SPCB keys. The size of the buffer needed is returned in <i>pulNumSPCBKeys</i> .
ERROR_INVALID_BLOCK	Invalid pointer.
FAILURE	Stream handler-specific error return code.

SpiEnumerateProtocols - Parameters

hid ([HID](#)) - input
ID (returned by [SpiGetHandler](#)) whose protocols are to be queried.

paSPCBKeys ([PVOID](#)) - in/out
Address of an array of SPCB keys to be filled in that identify the protocols supported by the specified stream handler.

pulNumSPCBKeys ([PULONG](#)) - in/out
Pointer to the number of entries on input. Upon return, contains the number actually returned. If more protocols are available than there are entries in the array, this function returns ERROR_INVALID_BUFFER_SIZE, and this parameter contains the number of array entries needed.

rc ([ULONG](#)) - returns
Return codes. Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_BUFFER_SIZE	Array passed is too small to hold all the SPCB keys. The size of the buffer needed is returned in <i>pulNumSPCBKeys</i> .
ERROR_INVALID_BLOCK	Invalid pointer.
FAILURE	Stream handler-specific error return code.

SpiEnumerateProtocols - Remarks

Use the SpiEnumerateProtocols function to determine which data types a given stream handler can process.

SpiEnumerateProtocols - Related Functions

- [SpiGetProtocol](#)
 - [SpiInstallProtocol](#)
 - [SpiGetHandler](#)
 - [SpiEnumerateHandlers](#)
-

SpiEnumerateProtocols - Related Messages

- [SHC_ENUMERATE_PROTOCOLS](#)
-

SpiEnumerateProtocols - Example Code

The following code illustrates how to return a list of stream protocol keys for the specified stream handler.

```
#include      "os2.h"
#include      "os2me.h"

ULONG        ulRC;                /* Error return code */
HID          hidTarget;           /* Target handler ID */
ULONG        ulNumSPCBKeys = 20; /* Number of entries */
SPCBKEY      aspcbkey[20]; /* Array of SPCB keys */

        .
        .
        .

/*-----*/
/*  Get the stream handler ID (hidSource) for the file      */
/*  stream handler using SpiGetHandler.                    */
/*-----*/

        .
        .
        .

/*-----*/
/*  Get a list of protocols for the digital audio stream   */
/*  handler.                                                */
/*-----*/
if (ulRC = SpiEnumerateProtocols( hidTarget, &aspcbkey, &ulNumSPCBKeys))
    return(ulRC); /* error! */
```

SpiEnumerateProtocols - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)

[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiGetHandler

SpiGetHandler - Syntax

This function returns the handler IDs for the specified stream handler.

```
#include <os2.h>

PSZ      pszHName; /* ASCIIZ name address. */
HID      *phidSrc; /* Stream handler ID. */
HID      *phidTgt; /* Stream handler ID. */
ULONG    rc;

rc = SpiGetHandler(pszHName, phidSrc, phidTgt);
```

SpiGetHandler Parameter - pszHName

pszHName ([PSZ](#)) - input

The address of the ASCIIZ name of the stream handler can be up to 9 characters (including the NULL).

SpiGetHandler Parameter - phidSrc

phidSrc ([HID *](#)) - output

Address of handler ID. This handler ID is used to identify the stream handler when it is used as a source stream handler.

SpiGetHandler Parameter - phidTgt

phidTgt ([HID *](#)) - output

Address of handler ID. This handler ID is used to identify the stream handler when it is used as a target stream handler.

SpiGetHandler Return Value - rc

rc (**ULONG**) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_HNDLR_NAME

Invalid stream-handler name.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_HNDLR_NOT_FOUND

The stream-handler name passed was not found in the system.

ERROR_HNDLR_NOT_IN_INI

The stream-handler name passed was not found in the SPI.INI file.

ERROR_INI_FILE

The SPI.INI file could not be found.

ERROR_LOADING_HNDLR

An error occurred loading and connecting the stream handler.

ERROR_READING_INI

An error occurred reading the SPI.INI file.

ERROR_INVALID_BLOCK

Invalid pointer.

SpiGetHandler - Parameters

pszHName (**PSZ**) - input

The address of the ASCIIZ name of the stream handler can be up to 9 characters (including the NULL).

phidSrc (**HID ***) - output

Address of handler ID. This handler ID is used to identify the stream handler when it is used as a source stream handler.

phidTgt (**HID ***) - output

Address of handler ID. This handler ID is used to identify the stream handler when it is used as a target stream handler.

rc (**ULONG**) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_HNDLR_NAME

Invalid stream-handler name.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_HNDLR_NOT_FOUND

The stream-handler name passed was not found in the system.

ERROR_HNDLR_NOT_IN_INI

The stream-handler name passed was not found in the SPI.INI file.

ERROR_INI_FILE

The SPI.INI file could not be found.

ERROR_LOADING_HNDLR

An error occurred loading and connecting the stream handler.

ERROR_READING_INI

An error occurred reading the SPI.INI file.

ERROR_INVALID_BLOCK

Invalid pointer.

SpiGetHandler - Remarks

If a DLL handler is not loaded, it will be loaded and connected to the Sync/Stream Manager at this point. All device driver stream handlers are loaded during CONFIG.SYS processing at operating-system startup.

Each stream handler can be a source or target or both for any stream instance. Therefore, a target [HID](#) for a stream is required if it can be a target, and a source [HID](#) is required if it can be a source. If the stream handler is NULL, the source [HID](#) and target [HID](#) are the same. If a stream handler can be used only as a target stream handler, the source [HID](#) returned is 0. If a stream handler can be used only as a source stream handler, the target [HID](#) returned is 0.

SpiGetHandler - Related Functions

- [SpiEnumerateHandlers](#)
- [SpiEnumerateProtocols](#)
- [SpiGetProtocol](#)
- [SpiInstallProtocol](#)

SpiGetHandler - Related Messages

- [SHC_INSTALL_PROTOCOL](#)

SpiGetHandler - Example Code

The following code illustrates how to return the handler IDs for the specified stream handler.

```
#include "os2.h"
#include "os2me.h"

ULONG ulRC;
HID hidSource,
hidTarget,
hidunused;

/* Error return code */
/* Source handler ID */
/* Target handler ID */
/* Dummy handler ID */
```

```

/*-----*/
/*  Get the stream handler ID for the file stream handler.  */
/*-----*/
if (ulRC = SpiGetHandler("FSSH",&hidSource, &hidunused))
    return(ulRC);                /* error! */

/*-----*/
/*  Get the stream handler ID for the digital audio stream handler*/
/*-----*/
if (ulRC = SpiGetHandler("AUDIOSH$",&hidunused, &hidTarget))
    return(ulRC);                /* error! */

```

SpiGetHandler - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiGetProtocol

SpiGetProtocol - Syntax

This function queries a stream handler for a specific stream protocol.

```

#include <os2.h>

HID      hid;          /* Stream handler ID. */
PSPCBKEY pspcbkey;     /* SPCB key. */
PSPCB    pspcb;        /* Memory block address. */
ULONG    rc;           /* Return codes indicating success or the type of failure: */

rc = SpiGetProtocol(hid, pspcbkey, pspcb);

```

SpiGetProtocol Parameter - hid

hid ([HID](#)) - input

ID of the stream handler from which to get the protocol.

SpiGetProtocol Parameter - pspcbkey

pspcbkey ([PSPCBKEY](#)) - input
Key (data type) that identifies the protocol to get.

SpiGetProtocol Parameter - pspcb

pspcb ([PSPCB](#)) - in/out
Address of the memory block to copy the SPCB to.

SpiGetProtocol Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_SPCBKEY	The protocol specified by the SPCB key was not recognized by the handler.
ERROR_INVALID_BLOCK	Invalid pointer.
FAILURE	Stream handler-specific error return code.

SpiGetProtocol - Parameters

hid ([HID](#)) - input
ID of the stream handler from which to get the protocol.

pspcbkey ([PSPCBKEY](#)) - input
Key (data type) that identifies the protocol to get.

pspcb ([PSPCB](#)) - in/out
Address of the memory block to copy the SPCB to.

rc ([ULONG](#)) - returns

Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_SPCBKEY	The protocol specified by the SPCB key was not recognized by the handler.
ERROR_INVALID_BLOCK	Invalid pointer.
FAILURE	Stream handler-specific error return code.

SpiGetProtocol - Remarks

This function is used to query a stream handler for a specified stream protocol. The *pspcbkey* indicates which stream protocol to return. The stream protocol can be modified and reinstalled into the stream handler for use in a stream instance.

SpiGetProtocol - Related Functions

- [SpiEnumerateProtocols](#)
- [SpiInstallProtocol](#)
- [SpiGetHandler](#)
- [SpiEnumerateHandlers](#)

SpiGetProtocol - Related Messages

- [SHC_GET_PROTOCOL](#)

SpiGetProtocol - Example Code

The following code illustrates how to query a stream handler for a specific stream protocol.

```
#include "os2.h"
#include "os2me.h"

ULONG ulRC;
HID hidTarget;
SPCBKEY spcbkey;
SPCB spcb;

/* Error return code */
/* Target handler ID */
/* Data type to stream */
/* Stream protocol */
```

```

        .
        .
        .
/*-----*/
/*  Get the stream handler ID (hidTarget) for the digital audio stream*/
/*  handler using SpiGetHandler.                                     */
/*-----*/
        .
        .
        .

/*-----*/
/*  Get the stream protocol control block (SPCB) for the given stream */
/*  handler and SPCB key.                                           */
/*-----*/
    spcbkey.ulDataType = DATATYPE_WAVEFORM;
    spcbkey.ulDataSubType = WAVE_FORMAT_4S16;
    spcbkey.ulIntKey = 0;

    if (ulRC = SpiGetProtocol( hidTarget, &spcbkey, &spcb))
        return(ulRC); /* error! */

```

SpiGetProtocol - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiGetTime

SpiGetTime - Syntax

This function queries the current stream time.

```

#include <os2.h>

HSTREAM    hstream; /* Stream handle. */
PMMTIME    pmmtime; /* Current stream time. */
ULONG      rc;      /* Return codes. */

rc = SpiGetTime(hstream, pmmtime);

```

SpiGetTime Parameter - hstream

hstream (**HSTREAM**) - input
Identifies the stream to perform the specified action.

SpiGetTime Parameter - pmmtime

pmmtime (**PMMTIME**) - in/out
Returns the current stream time in units of 1/3000th of a second.

SpiGetTime Return Value - rc

rc (**ULONG**) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_BLOCK	Invalid pointer.
ERROR_QUERY_STREAM_TIME	The source and target stream handlers cannot report stream time.
FAILURE	Stream handler-specific error return code.

SpiGetTime - Parameters

hstream (**HSTREAM**) - input
Identifies the stream to perform the specified action.

pmmtime (**PMMTIME**) - in/out
Returns the current stream time in units of 1/3000th of a second.

rc (**ULONG**) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.

Stream *time* is equivalent to stream *position*. This function is supported for any created stream, either active or stopped, until the stream is destroyed. See the [SpiSeekStream](#) function to change the current stream position.

- SpiEnableEvent
- SpiSeekStream

- SHC_GET_TIME

```
#include "os2.h"
#include "os2me.h"

ULONG ulRC; /* Error return code */
HSTREAM hStream; /* Stream handle */
MMTIME mmtime; /* Stream time */

.
.
.
/*-----*/
/* Create a data stream, associate the data with the stream, and start*/
/* streaming. (See SpiCreateStream, SpiAssociate, and SpiStartStream.)*/
/*-----*/

.
.
.
/*-----*/
/* Get stream time. */
/*-----*/
```

```
/*-----*/
if (ulRC = SpiGetTime (hStream, &mmtime ))
    return (ulRC);    /* error! */
```

SpiGetTime - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

SpiInstallProtocol

SpiInstallProtocol - Syntax

This function installs or removes a specific stream protocol for a stream handler.

```
#include <os2.h>

HID      hid;          /* Stream handler ID. */
PSPCBKEY pspcbkey;     /* SPCB key. */
PSPCB    pspcb;        /* SPCB address. */
ULONG    ulFlags;      /* Flags. */
ULONG    rc;           /* Return codes indicating success or the type of failure: */

rc = SpiInstallProtocol(hid, pspcbkey, pspcb,
    ulFlags);
```

SpiInstallProtocol Parameter - hid

hid ([HID](#)) - input

ID of the stream handler to which the protocol is to be set.

SpiInstallProtocol Parameter - pspcbkey

pspcbkey (**PSPCBKEY**) - input
Key (data type) that identifies the protocol.

SpilInstallProtocol Parameter - pspcb

pspcb (**PSPCB**) - input
Address of the SPCB to set.

SpilInstallProtocol Parameter - ulFlags

ulFlags (**ULONG**) - input
Contains one of the following flags:

SPI_INSTALL_PROTOCOL
Install this stream protocol in the specified stream handler. This is the default setting.

SPI_DEINSTALL_PROTOCOL
Remove this stream protocol from the specified stream handler.

SpilInstallProtocol Return Value - rc

rc (**ULONG**) - returns
Return codes indicating success or the type of failure:

NO_ERROR
Success.

ERROR_INVALID_FUNCTION
Invalid function requested.

ERROR_INVALID_HID
Invalid handler ID.

ERROR_INVALID_SPCBKEY
The protocol specified by SPCB was not recognized by the handler.

ERROR_INVALID_BLOCK
Invalid pointer.

ERROR_INVALID_BUFFER_SIZE
Invalid SPCB size.

ERROR_SPCBKEY_MISMATCH
The protocol specified by SPCB did not match the SPCBKEY.

FAILURE
Stream handler-specific error return code.

SpInstallProtocol - Parameters

hid ([HID](#)) - input

ID of the stream handler to which the protocol is to be set.

pspcbkey ([PSPCBKEY](#)) - input

Key (data type) that identifies the protocol.

pspcb ([PSPCB](#)) - input

Address of the SPCB to set.

ulFlags ([ULONG](#)) - input

Contains one of the following flags:

SPI_INSTALL_PROTOCOL

Install this stream protocol in the specified stream handler. This is the default setting.

SPI_DEINSTALL_PROTOCOL

Remove this stream protocol from the specified stream handler.

rc ([ULONG](#)) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_SPCBKEY

The protocol specified by SPCB was not recognized by the handler.

ERROR_INVALID_BLOCK

Invalid pointer.

ERROR_INVALID_BUFFER_SIZE

Invalid SPCB size.

ERROR_SPCBKEY_MISMATCH

The protocol specified by SPCB did not match the SPCBKEY.

FAILURE

Stream handler-specific error return code.

SpInstallProtocol - Remarks

The [SPCBKEY](#) field, *ullIntKey*, specifies which SPCB to change. In a multitasking environment, more than one application can be running and using the Sync/Stream Manager. To prevent one application from changing an SPCB in a stream handler that another application might be using, a unique *ullIntKey* must be used. For example, the stream handle or the device ID can be used as the *ullIntKey* to provide a unique value. This also enables any given application to define more than one SPCB for the same data type. There is no way to replace an SPCB that exists in a stream handler, but because adding and deleting are supported, the same function exists. An application cannot use this function to install or remove an SPCB with *ullIntKey* equal to 0.

Note: The *ullIntKey* field of [SPCBKEY](#) must match the *ullIntKey* field of *spcbkey* in the [SPCB](#) structure.

SpinstallProtocol - Related Functions

- [SpiEnumerateProtocols](#)
 - [SpiGetProtocol](#)
 - [SpiGetHandler](#)
 - [SpiEnumerateHandlers](#)
-

SpinstallProtocol - Related Messages

- [SHC_INSTALL_PROTOCOL](#)
-

SpinstallProtocol - Example Code

The following code illustrates how to install or remove a specific stream protocol for a stream handler.

```
#include "os2.h"
#include "os2me.h"

ULONG ulRC; /* Error return code */
HID hidTarget; /* Target handler ID */
SPCBKEY spcbkey; /* Data type to stream */
SPCB spcb;

.
.
.

/*-----*/
/* Get the stream handler ID (hidSource) for the file stream handler */
/* using SpiGetHandler. */
/*-----*/
.
.
.

/*-----*/
/* Get the stream protocol control block (SPCB) for the given stream */
/* handler and SPCB key. */
/*-----*/
spcbkey.ulDataType = DATATYPE_WAVEFORM;
spcbkey.ulDataSubType = WAVE_FORMAT_4S16;
spcbkey.ulIntKey = 0;

if (ulRC = SpiGetProtocol( hidTarget, &spcbkey, &spcb))
    return(ulRC); /* Error! */
.
.
.

/*-----*/
/* Install a stream protocol control block (SPCB) for the given */
/* handler and SPCB key. */
/*-----*/
spcbkey.ulIntKey = 1; /* Must be a unique number (non-zero) */
spcb.spcbkey.ulIntKey = 1; /* Ensure that the keys match. */
spcb.ulMaxBuf = 5; /* Change number of stream buffers. */
spcb.ulBufSize = 32768; /* Change size of buffers. */

if (ulRC = SpiInstallProtocol( hidTarget,
```



```

        &spcbkey,
        &spcb,
        SPI_INSTALL_PROTOCOL))
return(ulRC);
/* Error! */

```

SpiInstallProtocol - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiSeekStream

SpiSeekStream - Syntax

This function seeks to a specified point in the stream source object or sets the current stream time.

```

#include <os2.h>

HSTREAM    hstream;    /* Stream handle. */
ULONG      ulFlags;     /* Flags. */
LONG       lSeekPoint;  /* The point to which the stream attempts to seek. */
ULONG      rc;          /* Return codes indicating success or the type of failure: */

rc = SpiSeekStream(hstream, ulFlags, lSeekPoint);

```

SpiSeekStream Parameter - hstream

hstream ([HSTREAM](#)) - input
The stream that is to perform the specified action.

SpiSeekStream Parameter - ulFlags

ulFlags (**ULONG**) - input

Contains none or some of the following flags:

SPI_SEEK_ABSOLUTE

Seek from beginning of stream. This is the default setting.

SPI_SEEK_RELATIVE

Seek from current location.

SPI_SEEK_FROMEND

Seek from end of data object.

SPI_SEEK_SLAVES

Seek master as well as all slaves in the sync group. This flag can be used with any of the other flags.

SPI_SEEK_MMTIME

ISearchPoint is expressed in terms of MMTIME units (1/3000th of a second). This is the default setting.

SPI_SEEK_BYTES

ISearchPoint is expressed in bytes.

SPI_SEEK_IFRAME

Seek to nearest I-frame preceding the seek point.

SpiSeekStream Parameter - ISearchPoint

ISearchPoint (**LONG**) - input

The point to which the stream attempts to seek.

SpiSeekStream Return Value - rc

rc (**ULONG**) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_FLAG

The flag passed is invalid.

ERROR_INVALID_MMTIME

The specified MMTIME format is not understood by this stream.

ERROR_STREAM_NOT_STOP

The stream cannot perform the requested function unless the stream is stopped.

ERROR_DATA_ITEM_NOT_SPECIFIED

The data object is not specified.

ERROR_STREAM_NOT_SEEKABLE

Seek is an invalid request for this stream. (For example, seeking on a record does not make sense).

ERROR_STREAM_NOTMASTER

Seek stream with SPI_START_SLAVES flag was specified, but the stream handle passed was not a synchronization master stream.

ERROR_DATA_ITEM_NOT_SEEKABLE

Seek is an invalid request for the data object.

ERROR_NOT_SEEKABLE_BY_TIME

Cannot seek the data object using an MMTIME value.

ERROR_NOT_SEEKABLE_BY_BYTES

Cannot seek the data object using a byte value.

FAILURE

Stream handler-specific error return code.

SpiSeekStream - Parameters

hstream (**HSTREAM**) - input

The stream that is to perform the specified action.

ulFlags (**ULONG**) - input

Contains none or some of the following flags:

SPI_SEEK_ABSOLUTE

Seek from beginning of stream. This is the default setting.

SPI_SEEK_RELATIVE

Seek from current location.

SPI_SEEK_FROMEND

Seek from end of data object.

SPI_SEEK_SLAVES

Seek master as well as all slaves in the sync group. This flag can be used with any of the other flags.

SPI_SEEK_MMTIME

ISseekPoint is expressed in terms of MMTIME units (1/3000th of a second). This is the default setting.

SPI_SEEK_BYTES

ISseekPoint is expressed in bytes.

SPI_SEEK_IFRAME

Seek to nearest I-frame preceding the seek point.

ISseekPoint (**LONG**) - input

The point to which the stream attempts to seek.

rc (**ULONG**) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_FLAG

The flag passed is invalid.

ERROR_INVALID_MMTIME	The specified MMTIME format is not understood by this stream.
ERROR_STREAM_NOT_STOP	The stream cannot perform the requested function unless the stream is stopped.
ERROR_DATA_ITEM_NOT_SPECIFIED	The data object is not specified.
ERROR_STREAM_NOT_SEEKABLE	Seek is an invalid request for this stream. (For example, seeking on a record does not make sense).
ERROR_STREAM_NOTMASTER	Seek stream with SPI_START_SLAVES flag was specified, but the stream handle passed was not a synchronization master stream.
ERROR_DATA_ITEM_NOT_SEEKABLE	Seek is an invalid request for the data object.
ERROR_NOT_SEEKABLE_BY_TIME	Cannot seek the data object using an MMTIME value.
ERROR_NOT_SEEKABLE_BY_BYTES	Cannot seek the data object using a byte value.
FAILURE	Stream handler-specific error return code.

SpiSeekStream - Remarks

Depending on the type of object associated to the stream as a source or target, this function directs the appropriate stream handler to relocate the *current stream index*, or position, to the specified location. The stream position is equivalent to stream time. See [SpiGetTime](#) to query the current stream time of a stream instance. This function can be used to reposition the stream forward or backward from its current position. A negative *lSeekPoint* seeks backwards and a positive *lSeekPoint* seeks forward.

This function can be performed only on a newly created, not-started stream; a stopped stream (discard or flush stop); or after the end of stream notification. This function is not valid after a pause or if the stream is in a prerolled or prerolling state.

See the documentation for each stream handler for any restrictions to the seek function.

SpiSeekStream - Related Functions

- [SpiCreateStream](#)
- [SpiStartStream](#)
- [SpiEnableSync](#)
- [SpiGetTime](#)

SpiSeekStream - Related Messages

- [SHC_SEEK](#)

SpiSeekStream - Example Code

The following code illustrates how to seek a specified point in the stream source object or set the current stream time.

```
#include      "os2.h"
#include      "os2me.h"
#include      "stdio.h"

ULONG        ulRC;                        /* Error return code */
HSTREAM      hStream;                    /* Stream handle */
LONG         lSeekPoint;                 /* Position to seek to */
.
.
.
/*-----*/
/* Create a data stream, associate the data with the stream, and start*/
/* streaming. (See SpiCreateStream, SpiAssociate, and SpiStartStream. */
/*-----*/
.
.
.
/*-----*/
/* Seek to a position within the data stream. */
/*-----*/
lSeekPoint = 900000;                     /* Seek to 5-minute position */

if (ulRC = SpiSeekStream(hStream, SPI_SEEK_ABSOLUTE, lSeekPoint))
    return (ulRC);    /* Error! */
```

SpiSeekStream - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

SpiSendMsg

SpiSendMsg - Syntax

This function sends a stream-handler-specific message to a stream handler.

```
#include <os2.h>

HSTREAM      hstream;    /* Stream handle. */
```

```

HID      hid;          /* Stream handler ID. */
ULONG    ulMsgType;    /* Message type. */
PVOID     pMsg;        /* Pointer to control block. */
ULONG    rc;           /* Return codes. */

rc = SpiSendMsg(hstream, hid, ulMsgType, pMsg);

```

SpiSendMsg Parameter - hstream

hstream ([HSTREAM](#)) - input
The stream instance of the stream handler that will receive the message.

SpiSendMsg Parameter - hid

hid ([HID](#)) - input
ID of the stream handler to which the message should be sent.

SpiSendMsg Parameter - ulMsgType

ulMsgType ([ULONG](#)) - input
The stream-handler-specific message type.

SHC_REPORT_INT
pMsg is a pointer to [MSG_REPORTINT](#).

SHC_REPORT_EVENT
pMsg is a pointer to [MSG_REPORTEVENT](#).

SpiSendMsg Parameter - pMsg

pMsg ([PVOID](#)) - input
Pointer to a specific control block according to *ulMsgType*. The first field in the control block must be the length of the structure. See [MSG_COMMON](#).

SpiSendMsg Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
FAILURE	Stream handler-specific error return code.

SpiSendMessage - Parameters

hstream ([HSTREAM](#)) - input
The stream instance of the stream handler that will receive the message.

hid ([HID](#)) - input
ID of the stream handler to which the message should be sent.

ulMsgType ([ULONG](#)) - input
The stream-handler-specific message type.

SHC_REPORT_INT	<i>pMsg</i> is a pointer to MSG_REPORTINT .
SHC_REPORT_EVENT	<i>pMsg</i> is a pointer to MSG_REPORTEVENT .

pMsg ([PVOID](#)) - input
Pointer to a specific control block according to *ulMsgType*. The first field in the control block must be the length of the structure. See [MSG_COMMON](#).

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
FAILURE	Stream handler-specific error return code.

SpiSendMessage - Remarks

This function allows an application to directly communicate with a stream handler. It is used to pass information during active streaming. It is similar to the [SHC_ASSOCIATE](#) message, but it does not associate an object with a stream, and is usable when the stream is active. The message control block is stream handler specific and the interface must be provided by the stream handler.

SpiSendMsg - Related Functions

- [SpiAssociate](#)

SpiSendMsg - Related Messages

- [SHC_SENDMSG](#)

SpiSendMsg - Example Code

The following code illustrates how to seek a specified point in the stream source object or set the current stream time.

```
#define          INCL_ERRORS
#include         "os2.h"
#include         "os2me.h"
#include         "mcd.h"
#include         "audio.h"

ULONG          rc;                      /* error return code */
HID            hidSource;                /* source handler ID */
HSTREAM        hStream;                 /* stream handle */
ULONG          ulMsgType;                /* type of message */
MSG_COMMON     pMsg; /* message control block*/

        .
        .
        .

/*-----*/
/*  Create a data stream from a source stream handler to a target */
/*  stream handler.  (See SpiCreateStream). */
/*-----*/

        .
        .
        .

/*-----*/
/*  Fill in message from a source stream handler to a target handler*/
/*  */
/*-----*/

ulMsgType = STREAM_HANDLER_SPECIFIC_MESSAGE_NUMBER;
pMsg->ulMsgLen = sizeof(STREAM_HANDLER_SPECIFIC_MESSAGE_CONTROL_BLOCK);

/*-----*/
/*  Send a message to the source stream handler. */
/*  */
/*-----*/
if (rc = SpiSendMsg(hStream, hidSource, ulMsgType, pMsg, return)
    return (rc);
```

SpiSendMsg - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

SpiStartStream

SpiStartStream - Syntax

This function starts data streaming for a stream instance or a group of streams.

```
#include <os2.h>

HSTREAM    hstream; /* Stream handle. */
ULONG      ulFlags; /* Flags. */
ULONG      rc;       /* Return codes indicating success or the type of failure: */

rc = SpiStartStream(hstream, ulFlags);
```

SpiStartStream Parameter - hstream

hstream ([HSTREAM](#)) - input

The stream that is to perform the specified action.

SpiStartStream Parameter - ulFlags

ulFlags ([ULONG](#)) - input

Contains none or some of the following flags:

SPI_START_STREAM

Start master stream only. This is the default setting.

SPI_START_SLAVES

Start master stream and any slave streams in the same sync group. If this flag is set, the *hstream* must be the stream handle of a master stream.

SPI_START_PREROLL

Preroll the stream; that is, fill all the stream buffers with data. Once this is done, the stream is in a prerolled state,

which enables the stream to be started more quickly. This is an asynchronous function.

SpiStartStream Return Value - rc

rc (**ULONG**) - returns

Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_FLAG	The flag passed is invalid.
ERROR_STREAM_NOTMASTER	Start stream with SPI_START_SLAVES flag was specified, but the stream handle passed was not a synchronization master stream.
ERROR_START_STREAM	There was an error starting the stream.
ERROR_DATA_ITEM_NOT_SPECIFIED	The data object was not specified.
ERROR_DEVICE_NOT_FOUND	The device was not found.
FAILURE	Stream handler-specific error return code.

SpiStartStream - Parameters

hstream (**HSTREAM**) - input

The stream that is to perform the specified action.

uiFlags (**ULONG**) - input

Contains none or some of the following flags:

SPI_START_STREAM	Start master stream only. This is the default setting.
SPI_START_SLAVES	Start master stream and any slave streams in the same sync group. If this flag is set, the <i>hstream</i> must be the stream handle of a master stream.
SPI_START_PREROLL	Preroll the stream; that is, fill all the stream buffers with data. Once this is done, the stream is in a prerolled state, which enables the stream to be started more quickly. This is an asynchronous function.

rc (**ULONG**) - returns

Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_FLAG	The flag passed is invalid.
ERROR_STREAM_NOTMASTER	

	Start stream with SPI_START_SLAVES flag was specified, but the stream handle passed was not a synchronization master stream.
ERROR_START_STREAM	There was an error starting the stream.
ERROR_DATA_ITEM_NOT_SPECIFIED	The data object was not specified.
ERROR_DEVICE_NOT_FOUND	The device was not found.
FAILURE	Stream handler-specific error return code.

SpiStartStream - Remarks

The start or preroll start can be issued only for a stream that is in an idle state; that is, the stream is not started or is paused, stopped, or prerolled). Any attempt to start a stream that is already started or in the process of prerolling results in ERROR_START_STREAM.

If multiple streams are involved when the synchronization master stream is started, all slave streams also can be started by using the SPI_START_SLAVES flag. This synchronization relationship can be set up using [SpiEnableSync](#) after all streams are created.

The active stream protocol governs the amount of data buffered and the rate of data flow maintained. Both source and target stream handlers are controlled by the stream protocol (SPCB). Both handlers contribute to the contents of the SPCB at stream creation. Stream events, such as cue points, are detected by the appropriate stream handler, and an event notification is sent to the process that owns the stream.

This function also enables a stream or group of streams to be prerolled. A preroll start enables the source stream handlers to start and fill the buffers. An EVENT_SYNC_PREROLLED event is generated by the Sync/Stream Manager when the stream is prerolled. This enables an application to start the stream or streams for better real-time response and initial synchronization of the streams. For group prerolling, only one EVENT_SYNC_PREROLLED event message is sent when all of the streams in the group have prerolled.

Once a stream is started, errors are reported to the application through the event routine as an EVENT_ERROR event. For example, an SpiStartStream function might return with a NO_ERROR return code but fail to start and return EVENT_ERROR asynchronously. This function should not be used during exit-list processing because it is asynchronous and the event for completion of the function will never be sent.

SpiStartStream - Related Functions

- [SpiAssociate](#)
- [SpiCreateStream](#)
- [SpiStopStream](#)
- [SpiEnableSync](#)

SpiStartStream - Related Messages

- [SHC_START](#)

SpiStartStream - Example Code

The following code illustrates how to start data streaming for a stream instance or a group of streams.

```

#include      "os2.h"
#include      "os2me.h"

        ULONG      ulRC;                /* Error return code */
        HSTREAM     hStream;            /* Stream handle */

        .
        .
        .
/*-----*/
/*  Create a data stream and associate the data with the stream.*/
/*  (See SpiCreateStream and SpiAssociate.)                      */
/*                                                                */
/*-----*/
        .
        .
        .
/*-----*/
/*  Start the streaming.                                          */
/*-----*/
        if (ulRC = SpiStartStream(hStream, SPI_START_STREAM))
            return (ulRC);    /* Error! */

```

SpiStartStream - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Related Messages](#)

[Glossary](#)

SpiStopStream

SpiStopStream - Syntax

This function stops data streaming for a stream instance or a group of streams.

```

#include <os2.h>

HSTREAM     hstream;    /* Stream handle. */
ULONG       ulFlags;    /* Flags. */
ULONG       rc;         /* Return codes indicating success or the type of failure: */

rc = SpiStopStream(hstream, ulFlags);

```

SpiStopStream Parameter - hstream

hstream (**HSTREAM**) - input
The stream that is to perform the specified action.

SpiStopStream Parameter - ulFlags

ulFlags (**ULONG**) - input
Contains none or some of the following flags:

SPI_STOP_STREAM
Pause the stream. No data is lost. This is the default setting.

SPI_STOP_SLAVES
Stop master stream and any slave streams in the same sync group. If this flag is set, the *hstream* must be the stream handle of a master stream.

SPI_STOP_FLUSH
Continue to stream data until all stream buffers are empty. No new full buffers are accepted from the source stream handler. This is an asynchronous function.

SPI_STOP_DISCARD
Stop the stream and discard all stream buffers. This is an asynchronous function.

SpiStopStream Return Value - rc

rc (**ULONG**) - returns
Return codes indicating success or the type of failure:

NO_ERROR
Success.

ERROR_INVALID_FUNCTION
Invalid function requested.

ERROR_INVALID_STREAM
Invalid stream handle.

ERROR_INVALID_FLAG
The flag passed is invalid.

ERROR_STREAM_NOTMASTER
Stop stream with SPI_STOP_SLAVES flag was specified, but the stream handle passed was not a synchronization master stream.

ERROR_STREAM_STOP_PENDING
A stream-stop is already pending.

ERROR_STREAM_ALREADY_STOP
The stream has been stopped or has reached the end of the data and stopped.

ERROR_STREAM_NOT_STARTED
The stream is either in the process of prerolling or the stream has been prerolled and a flush stop is requested.

ERROR_STREAM_ALREADY_PAUSE
The stream is already stopped (pause).

FAILURE

Stream handler-specific error return code.

SpiStopStream - Parameters

hstream ([HSTREAM](#)) - input

The stream that is to perform the specified action.

ulFlags ([ULONG](#)) - input

Contains none or some of the following flags:

SPI_STOP_STREAM

Pause the stream. No data is lost. This is the default setting.

SPI_STOP_SLAVES

Stop master stream and any slave streams in the same sync group. If this flag is set, the *hstream* must be the stream handle of a master stream.

SPI_STOP_FLUSH

Continue to stream data until all stream buffers are empty. No new full buffers are accepted from the source stream handler. This is an asynchronous function.

SPI_STOP_DISCARD

Stop the stream and discard all stream buffers. This is an asynchronous function.

rc ([ULONG](#)) - returns

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_FLAG

The flag passed is invalid.

ERROR_STREAM_NOTMASTER

Stop stream with SPI_STOP_SLAVES flag was specified, but the stream handle passed was not a synchronization master stream.

ERROR_STREAM_STOP_PENDING

A stream-stop is already pending.

ERROR_STREAM_ALREADY_STOP

The stream has been stopped or has reached the end of the data and stopped.

ERROR_STREAM_NOT_STARTED

The stream is either in the process of prerolling or the stream has been prerolled and a flush stop is requested.

ERROR_STREAM_ALREADY_PAUSE

The stream is already stopped (pause).

FAILURE

Stream handler-specific error return code.

SpiStopStream - Remarks

Three possible types of stops can be issued: pause, discard, and flush. The pause stop immediately pauses the stream by sending an [SHC_STOP](#) call to both the source and target stream handlers. Both stream handlers must become idle and not request any more buffers from the Sync/Stream Manager. The Sync/Stream Manager will fail any "get buffer" requests, so that a stream handler cannot continue to stream. All stream data remains valid. The stream can be restarted by issuing [SpiStartStream](#).

The discard stop immediately stops the stream by sending an [SHC_STOP](#) call to both the source and target stream handlers. Both stream handlers must return all buffers to the Sync/Stream Manager. The Sync/Stream Manager fails any "get buffer" requests, so that a stream handler cannot continue to stream. Any data currently in the stream buffers is discarded. The stream can be restarted after a discard stop. The stream time is adjusted to the time at the point the stream was stopped.

The flush stop sends an [SHC_STOP](#) call to both the source and target stream handlers. The stream does not stop, however. The source stream handler must not request any more buffers from the Sync/Stream Manager. It must return all buffers to the Sync/Stream Manager. The Sync/Stream Manager fails any "get buffer" requests from the source stream handler, so that it cannot continue to fill buffers. The target stream handler continues to stream data until all buffers are emptied. The stream can be restarted, with no loss of data.

For discard and flush stops, the Sync/Stream Manager detects when the stream handlers have stopped and returned all buffers. At this point, the Sync/Stream Manager notifies the application with an `EVENT_STREAM_STOPPED` event message. In the case of a sync group, there will be one `EVENT_STREAM_STOPPED` event for each stream that is stopped. Any stream in the sync group that was previously stopped, either by an explicit [SpiStartStream](#) or by reaching the end of the stream, does not trigger an `EVENT_STREAM_STOPPED` event.

If a stream is paused and the application issues a discard stop, the stream buffers are discarded and the stream is put in a stopped state. If a stream is paused and a flush stop is issued, the remaining stream buffer data is transferred to the target stream handler. In other words, the stream begins *playing* again.

This function returns an error if a flush stop is issued for a prerolled stream that has not been started. A flush or pause stop is not valid while a stream is being prerolled. If a pause stop is issued to a prerolled stream, the stream is still considered to be prerolled and not paused.

Slave streams can be stopped independently of the master stream. When the slave stream stops, the sync relationship is not broken, but the slave becomes idle. The master stream and any other slave streams continue running. If the stopped slave is restarted, a new master-to-slave time offset is established, which the slave stream maintains until it is either stopped again, or the master stream is stopped. As a result, slave streams can regain synchronization at any asynchronous point in time, as in response to another stream event from the master stream, for example.

Slave-stream time and master-stream time are not always identical. Once a slave stream is stopped, the master stream time can continue to increment. When the slave stream is restarted, its stream time increments at the same rate as the master stream. Any slave stream in a synchronization relationship can be started arbitrarily or stopped without affecting the activity of the master stream or any of the slave streams. When the `SPI_STOP_SLAVES` flag is set, the master stream, as well as the slave streams, stops.

Typically, data streaming continues until the end of the stream. When this happens, the Sync/Stream Manager sends an `EVENT_EOS` event message to the application's event routine. In the case of a sync group, there is one `EVENT_EOS` message per stream.

Note: This function should not be used during exit-list processing because it is asynchronous and the event for completion of the function will never be sent.

SpiStopStream - Related Functions

- [SpiCreateStream](#)
- [SpiDestroyStream](#)
- [SpiStartStream](#)
- [SpiEnableSync](#)

SpiStopStream - Related Messages

- [SHC_STOP](#)

SpiStopStream - Example Code

The following code illustrates how to stop data streaming for a stream instance or a group of streams.

```
#include      "os2.h"
#include      "os2me.h"

    ULONG      ulRC;                      /* Error return code */
    HSTREAM     hStream;                  /* Stream handle */
    .
    .
    .

/*----- */
/* Create a data stream, associate the data with the stream and start */
/* streaming. (See SpiCreateStream, SpiAssociate, and SpiStartStream.)*/
/*----- */
    .
    .
    .

/*----- */
/*   Now stop the streaming.                                           */
/*----- */
if (ulRC = SpiStopStream(hStream, SPI_STOP_STREAM))
    return (ulRC);    /* Error! */
```

SpiStopStream - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Related Functions](#)
 - [Related Messages](#)
 - [Glossary](#)

Data Stream State Table

The following table identifies the data stream states during which a call to certain SPI functions returns an error to the application.

Key:

- E Function returns an error.
- OK Function can be called.
- N/A Not applicable.

SPI Function	Running State	Paused State	Stopped State	Flush Pending State	Discard Pending State	Destroy Pending State	Pre-rolling State	Pre-rolled State
SpiAssociate	E	E	OK	E	E	E	E	E
SpiEnumerateHandlers	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

SpiGetHandler	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SpiCreateStream	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SpiDestroyStream	OK	OK	OK	OK	OK	E	OK	OK
SpiGetTime	OK	OK	OK	OK	OK	E	OK	OK
SpiSeekStream	E	E	OK	E	E	E	E	E
SpiStartStream	E	OK	OK	E	E	E	E	OK
SpiStartStream (Preroll)	E	OK	OK	E	E	E	E	OK
SpiStopStream (Flush)	OK	OK	E	E	E	E	E	E
SpiStopStream (Discard)	OK	OK	E	E	E	E	OK	OK
SpiStopStream (Pause)	OK	E	E	E	E	E	E	OK
SpiEnableEvent	OK	OK	OK	OK	OK	E	OK	OK
SpiDisableEvent	OK	OK	OK	OK	OK	E	OK	OK
SpiEnableSync	OK	OK	OK	OK	OK	E	OK	OK
SpiDisableSync	OK	OK	OK	OK	OK	E	E	OK
SpiGetProtocol	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SpiEnumerateProtocols	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SpiInstallProtocol	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SpiDetermineSyncMaster	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

SMH Messages

The Sync/Stream Manager helper (SMH) messages, provided by the Sync/Stream Manager, are used by all stream handlers to do the following.

- Register with the Sync/Stream Manager.
- Report events and synchronization pulses to the Sync/Stream Manager.
- Request or return buffers to the Sync/Stream Manager.

The SMH messages are synchronous calls and available to both DLL stream handlers (as a DLL call) and to device driver stream handlers (as an IDC call).

These helper messages are provided through a single entry point, [SMHEntryPoint](#), which accepts a parameter structure as input. This permits the DLL and the device driver interfaces to the Sync/Stream Manager to be the same. For the DLL interface, all pointers are 0:32 linear; for the device driver interface, all are 16:16 pointers, which enables the current 16-bit device-driver model to be used for device driver stream handlers. The following table contains the message numbers for all Sync/Stream Manager helper messages. These numbers must be used in the *ulFunction* field, of the parameter structure passed in the call, to indicate which message is being requested by the stream handler.

The following table lists the SMH messages.

Message Number	Message	Description
1L	SMH_DEREGISTER	Disconnects a stream handler from the Sync/Stream Manager.

4L	SMH_LOCKMEM	Locks a memory object. (Available at ring 3 only.)
3L	SMH_NOTIFY	Notifies the Sync/Stream Manager of a stream buffer request.
0L	SMH_REGISTER	Registers a stream handler with the Sync/Stream Manager.
2L	SMH_REPORTEVENT	Reports events or sync pulses to the Sync/Stream Manager.

All of these messages are available to DLL stream handlers, as well as to device driver stream handlers. The Sync/Stream Manager device driver IDC entry point can be acquired by calling the ATTACHDD DevHelp function when the device driver is initialized; however, the IDC entry point cannot be used until after the device drive is initialized. As a result, it is impossible for a device driver stream handler to issue [SMH_REGISTER](#) through the IDC interface. The Sync/Stream Manager provides the [SMH_REGISTER](#) through a call to DosDevIOctl, so that [SMH_REGISTER](#) can be issued during stream handler initialization. The following information is required in the [SMH_REGISTER](#) DosDevIOctl call:

The SMH_REGISTER DosDevIOctl interface:

Device driver name	"SSM\$"
Category	0x0081
SMH_REGISTER message	0x0040
Parameter packet	Same as IDC interface
Data packet	NONE

SMHEntryPoint

SMHEntryPoint - Syntax

This function enables DLL and device driver stream handlers to interface with the Sync/Stream Manager (SSM).

```
#include <os2.h>

PVOID    pParmIn; /* Pointer to SMH_COMMON. */
ULONG    rc;      /* Return codes. */

rc = SMHEntryPoint(pParmIn);
```

SMHEntryPoint Parameter - pParmIn

pParmIn ([PVOID](#)) - input
Pointer to an SMH message-specific input parameter block. See [SMH_COMMON](#).

SMHEntryPoint Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or failure:

NO_ERROR

Successful.

ERROR_INVALID_FUNCTION

Invalid function requested.

FAILURE

An SMH message-specific error return code.

SMHEntryPoint - Parameters

pParmln ([PVOID](#)) - input

Pointer to an SMH message-specific input parameter block. See [SMH_COMMON](#).

rc ([ULONG](#)) - returns

Return codes indicating success or failure:

NO_ERROR

Successful.

ERROR_INVALID_FUNCTION

Invalid function requested.

FAILURE

An SMH message-specific error return code.

SMHEntryPoint - Remarks

The Stream Manager Helper (SMH) messages are exported for use by both device driver and DLL stream handlers through the SMHEntryPoint. The stream handlers use these helper messages to register with the Sync/Stream Manager, report events and synchronization cues, and request or return buffers. They are synchronous calls and are available to DLL stream handlers as a DLL call (device driver stream handlers as an IDC call).

In addition, device drivers stream handlers send SMH messages to the Sync/Stream Manager (SSM) through the SMHEntryPoint. For device driver stream handlers, this interface is created using the standard inter-device driver communication (IDC) approach, established by the AttachDD DevHelp function during initialization processing.

SMHEntryPoint - Example Code

The following code illustrates how to enable DLL and device driver stream handlers to interface with the Sync/Stream Manager.

```
#include "os2.h"
#include "os2me.h"
```

```

ULONG          ulRC;                /* Error return code          */
PARM_LOCKM     parm_lockm;         /* Lock memory parameter block */
LOCKH          lockh;              /* Lock handle                */
PVOID          pMem;               /* Pointer to memory object    */
PSMHFN         SMHEntryPoint;      /* Pointer to SMH entry point  */

        .
        .
        .
/*-----
*   Allocate memory and save address in pMem.
*-----
        .
        .
        .

*-----
*   Lock the allocated memory.
*-----*/
parm_lockm.ulFunction = SMH_LOCKMEM;
parm_lockm.pBuffer = pMem;
parm_lockm.ulBufSize = 4096;
parm_lockm.plockh = &lockh;
parm_lockm.ulFlags = SSM_LOCKMEM | SSM_CONTIGLOCK;

if (ulRC = SMHEntryPoint (&parm_lockm))
    return(ulRC);                /* Error! */

```

SMHEntryPoint - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

SMH_DEREGISTER

SMH_DEREGISTER Parameter - pParmln

pParmln ([PPARM_DEREG](#))

A pointer to a [PARM_DEREG](#) data structure.

SMH_DEREGISTER Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

<code>NO_ERROR</code>	Function performed.
<code>ERROR_INVALID_FUNCTION</code>	Invalid function requested.
<code>ERROR_HNDLR_NOT_FOUND</code>	Stream-handler registration information was not found.
<code>ERROR_INVALID_HNDLR_NAME</code>	Invalid stream-handler name.
<code>ERROR_INVALID_BLOCK</code>	Invalid pointer.

SMH_DEREGISTER - Description

This message disconnects a stream handler from the Sync/Stream Manager.

pParmIn ([PPARM_DEREG](#))

A pointer to a [PARM_DEREG](#) data structure.

rc ([ULONG](#))

Return codes indicating success or the type of failure:

<code>NO_ERROR</code>	Function performed.
<code>ERROR_INVALID_FUNCTION</code>	Invalid function requested.
<code>ERROR_HNDLR_NOT_FOUND</code>	Stream-handler registration information was not found.
<code>ERROR_INVALID_HNDLR_NAME</code>	Invalid stream-handler name.
<code>ERROR_INVALID_BLOCK</code>	Invalid pointer.

SMH_DEREGISTER - Remarks

A stream handler can use this message to deregister itself with the Sync/Stream Manager. For example, if a stream handler DLL comes across an error condition that requires termination of its worker thread, the stream handler must deregister with the Sync/Stream Manager because it no longer can support calls. This is used mostly by the device driver stream handler in the case where the device driver receives an error and is no longer callable.

SMH_DEREGISTER - Example Code

The following code illustrates how to disconnect a stream handler from the Sync/Stream Manager.

```
#include      "os2.h"
#include      "os2me.h"

#define      RegName      "C:\PATH\TESTSH"  /* Handler name and path */

      ULONG      ulRC;                      /* Error return code */
      PARM_DEREG      parm_dereg;          /* Deregister parameter block */
      PSMHFN      SMHEntryPoint;          /* Pointer to SMH entry point */

/*-----*/
/* Deregister a stream handler.          */
/*-----*/
      parm_dereg.ulFunction = SMH_DEREGISTER; /* Set function */
      parm_dereg.pszSHName = (PSZ) RegName; /* Set handler name */

      if (ulRC = SMHEntryPoint (&parm_dereg))
          return(ulRC); /* Error! */
```

SMH_DEREGISTER - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

SMH_LOCKMEM

SMH_LOCKMEM Parameter - pParmln

pParmln ([PPARM_LOCKM](#))

A pointer to a [PARM_LOCKM](#) data structure.

SMH_LOCKMEM Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Function performed.

ERROR_INVALID_FUNCTION
Invalid function requested.

ERROR_INVALID_BLOCK
Invalid pointer.

SMH_LOCKMEM - Description

This message locks a memory object (available at Ring 3 only).

pParmIn ([PPARM_LOCKM](#))
A pointer to a [PARM_LOCKM](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR
Function performed.

ERROR_INVALID_FUNCTION
Invalid function requested.

ERROR_INVALID_BLOCK
Invalid pointer.

SMH_LOCKMEM - Remarks

This message is available for Ring 3 DLL stream handlers to lock memory objects so they cannot be paged out. Currently, the buffering algorithms and buffer management code of the Sync/Stream Manager locks down stream buffers. Therefore, this message should be used only to lock down code segments or to lock down stack segments so the stream handler does not get swapped out during streaming. You also might use this message in the case of user-provided buffers. For example, if there were a particular type of stream created where the stream handler DLL or source stream handler actually provided application memory buffers, you could use locked memory to lock down buffers that the handler passed to the Sync/Stream Manager. Typically, the Sync/Stream Manager attempts to lock down the buffers anyway.

SMH_LOCKMEM - Example Code

The following code illustrates how to lock a memory object.

```
#include "os2.h"
#include "os2me.h"

ULONG      ulRC;                /* Error return code */
PARM_LOCKM parm_lockm;         /* Lock memory parameter block */
LOCKH      lockh;              /* Lock handle */
PVOID      pMem;               /* Pointer to memory object */
PSMHFN     SMHEntryPoint;      /* Pointer to SMH entry point */

:
```

```

/*-----
* Allocate memory and save address in pMem.
*-----
.
.
.

*-----
* Lock the allocated memory.
*----- */
parm_lockm.ulFunction = SMH_LOCKMEM;
parm_lockm.pBuffer = pMem;
parm_lockm.ulBufSize = 4096;
parm_lockm.plockh = &lockh;
parm_lockm.ulFlags = SSM_LOCKMEM | SSM_CONTIGLOCK;

if (ulRC = SMHEntryPoint (&parm_lockm))
    return(ulRC); /* Error! */

```

SMH_LOCKMEM - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

SMH_NOTIFY

SMH_NOTIFY Parameter - pParmIn

pParmIn ([PPARM_NOTIFY](#))

A pointer to a [PARM_NOTIFY](#) data structure. The *pGetBufTab* field points to the [SRCBUFTAB](#) data structure. The *pRetBufTab* field points to the [TGTBUFTAB](#) data structure.

SMH_NOTIFY Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Message performed.

ERROR_BUFFER_NOT_AVAILABLE

No buffers available for this request. Possible reasons are:

- For multiple-buffer requests, part of the request might have been satisfied. The *ulGetNumEntries* or *ulRetNumEntries* fields will contain the number of buffers that were successful.
- A stream is pending completion of a destroy or stop request.
- The stream is stopped.
- End of stream has been reached.

ERROR_TOO_MANY_BUFFERS

For a BUF_GIVE_BUF request, too many buffers have been given to the Sync/Stream Manager. Increase the *ulMaxBuf* field in the [SPCB](#) to be able to specify more buffers.

ERROR_TOO_MANY_RECORDS

For a BUF_RETURNFULL+BUF_RECORDS request, not enough resources are available to handle the complete request. The *ulRetNumEntries* field will contain the number of entries accepted on the return. The stream handler must block itself and wait for resources. This is similar to the ERROR_BUFFER_NOT_AVAILABLE response.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_FLAG

Invalid input flag value.

ERROR_LOCKING_BUFFER

For a BUF_GIVEBUF request, there was a problem locking the buffer. The caller must block until [SHC_START](#) is received from the Sync/Stream Manager and then try the request again.

ERROR_UNLOCKING_BUFFER

For a BUF_RETURNEMPTY request, there was a problem unlocking a user-provided buffer.

ERROR_INVALID_BLOCK

Invalid pointer.

ERROR_INVALID_BUFFER_SIZE

Invalid buffer size in a Return Full or Give Buffer request.

ERROR_INVALID_BUFFER_RETURNED

Invalid buffer was returned. This might happen if a stream handler attempts to return the same buffer more than once.

ERROR_INVALID_RECORD_RETURNED

The record returned does not belong to the buffer specified on a Return Full request.

ERROR_STREAM_NOT_OWNER

A source stream handler tried to Get Empty a buffer from a stream that is using the stream's buffers (split streams only).

ERROR_STREAM_NOT_ACTIVE

A stream is pending completion of a destroy or stop request, or the stream is stopped.

ERROR_INVALID_PARAMETER

Invalid parameter.

SMH_NOTIFY - Description

This message notifies the Sync/Stream Manager of a stream buffer request.

pParmln (PPARM_NOTIFY)

A pointer to a [PARM_NOTIFY](#) data structure. The *pGetBufTab* field points to the [SRCBUFTAB](#) data structure. The *pRetBufTab* field points to the [TGTBUFTAB](#) data structure.

rc (ULONG)

Return codes indicating success or the type of failure:

NO_ERROR

Message performed.

ERROR_BUFFER_NOT_AVAILABLE

No buffers available for this request. Possible reasons are:

- For multiple-buffer requests, part of the request might have been satisfied. The *ulGetNumEntries* or *ulRetNumEntries* fields will contain the number of buffers that were successful.
- A stream is pending completion of a destroy or stop request.
- The stream is stopped.
- End of stream has been reached.

ERROR_TOO_MANY_BUFFERS

For a BUF_GIVE_BUF request, too many buffers have been given to the Sync/Stream Manager. Increase the *ulMaxBuf* field in the [SPCB](#) to be able to specify more buffers.

ERROR_TOO_MANY_RECORDS

For a BUF_RETURNFULL+BUF_RECORDS request, not enough resources are available to handle the complete request. The *ulRetNumEntries* field will contain the number of entries accepted on the return. The stream handler must block itself and wait for resources. This is similar to the ERROR_BUFFER_NOT_AVAILABLE response.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_FLAG

Invalid input flag value.

ERROR_LOCKING_BUFFER

For a BUF_GIVEBUF request, there was a problem locking the buffer. The caller must block until [SHC_START](#) is received from the Sync/Stream Manager and then try the request again.

ERROR_UNLOCKING_BUFFER

For a BUF_RETURNEMPTY request, there was a problem unlocking a user-provided buffer.

ERROR_INVALID_BLOCK

Invalid pointer.

ERROR_INVALID_BUFFER_SIZE

Invalid buffer size in a Return Full or Give Buffer request.

ERROR_INVALID_BUFFER_RETURNED

Invalid buffer was returned. This might happen if a stream handler attempts to return the same buffer more than once.

ERROR_INVALID_RECORD_RETURNED

The record returned does not belong to the buffer specified on a Return Full request.

ERROR_STREAM_NOT_OWNER

A source stream handler tried to Get Empty a buffer from a stream that is using the stream's buffers (split streams only).

ERROR_STREAM_NOT_ACTIVE

A stream is pending completion of a destroy or stop request, or the stream is stopped.

ERROR_INVALID_PARAMETER

Invalid parameter.

SMH_NOTIFY - Remarks

This message is used to get or return an empty or full buffer, depending on whether the stream handler is a target or source of data. The message can return or request not only the next available buffer, but also the next *n* buffers. For example, a target stream handler can request the next 3 available full buffers, instead of the next available buffer, if it needs to speed up its consumption of buffers.

The target stream handler should return the empty buffers in the reverse order in which they were retrieved from the Sync/Stream Manager. The target stream handler must be able to accept a NULL buffer pointer on a BUF_GETFULL request because the BUF_EOS flag may be set, but there may not be any buffers left to consume.

All buffers allocated by the Sync/Stream Manager are fixed and cannot be swapped.

The Sync/Stream Manager provides the ability for a source stream handler to associate one cue point with a buffer that it passes to the Sync/Stream Manager. This buffer and attribute remain in the data stream until the target stream handler requests it. The attribute for the buffer is passed with the buffer to the target. See the *ulMessageParm* and *mmtimeOffset* parameters for a description of the buffer attribute. The target stream handler then must report an EVENT_PLAYLISTCUEPOINT event, at the appropriate time, with the buffer attribute as the message parameter (the *ulMessageParm* field in the [PLAYL_EVCB](#) structure).

This source stream handler feature of the Sync/Stream Manager supports the memory-playlist ability to send a cue-point event as close to the actual time it occurs as possible. For this to work, it must be the target stream handler that detects the cue-point event and reports it to the application media control driver. If the system-memory stream handler is the source stream handler, it must notify the target stream handler to report a cue-point event.

The contents of the *ulMessageParm* parameter are defined by the user application. For example, if the system-memory stream handler is the source stream handler, and it receives a playlist from the user application, the playlist CUEPOINT instruction defines the contents of the buffer attribute passed to the target stream handler. The target stream handler then reports the event and passes the buffer attribute as an event parameter.

For split streams, a buffer can be divided into multiple variable-length records. This is useful for streaming interleaved data from one source stream handler, in multiple streams, to one or more target stream handlers, where each stream represents a different data type.

The source stream handler requests one buffer at a time from the Sync/Stream Manager and returns filled records to the Sync/Stream Manager. The source stream handler specifies which stream the records will go in. Each stream goes to a different target stream handler. The *pRecord* field of the pointer table are not used for normal non-split streams. The source stream handler can return multiple records of a specific data type to a stream, but these all must be contained within the same buffer. The last record of each buffer is indicated by using the BUF_LASTRECORD flag; it is only sent to the stream that receives the last record for a buffer. The BUF_EOS flag, on the other hand, is sent to all streams when the last buffer or record is returned to the Sync/Stream Manager.

The SMH_NOTIFY message also can accept user-provided buffers into a stream to be passed directly from the source stream handler to the target stream handler. This is accomplished with the BUF_GIVEBUF flag, which enables the source stream handler to *give* a buffer and length to the Sync/Stream Manager eventually to be passed to the target stream handler for a BUF_GETFULL request. Each buffer given to the Sync/Stream Manager is used once. The BUF_GIVEBUF is only available on DLL SMH_NOTIFY calls. The Sync/Stream Manager returns ERROR_TOO_MANY_BUFFERS to the source stream handler if it attempts to give too many buffers. The BUF_GIVEBUF feature is provided primarily for the memory stream handler to be able to stream data from an application's memory.

For DLL stream handlers, all pointers returned are 32-bit flat process linear addresses. For device driver stream handlers, pointers returned in an SMH_NOTIFY message are either 32-bit, flat, global linear addresses, physical addresses, or *GDT* selector and offset addresses. The latter is the default. The others are selectable for each SMH_NOTIFY message.

Note: The physical addresses are pointers to physically contiguous memory, not a page table. The SPCBBUF_NONCONTIGUOUS bit must not be set to On in the [SPCB](#) to get a physical address from the Sync/Stream Manager. The Sync/Stream Manager fails requests for physical addresses if this bit is set.

Valid Source SH (PRODUCER) Request Combinations to SMH_NOTIFY:

Any combination of BUF_GETEMPTY and BUF_RETURNFULL can be used for the same SMH_NOTIFY function.

Either BUF_LINEAR or BUF_PHYSICAL can be used with any combination of BUF_GETEMPTY, BUF_RETURNFULL, BUF_GIVEBUF, BUF_GDT, and BUF_EXTENDEDPTR. BUF_LINEAR and BUF_PHYSICAL are valid only for device driver stream handlers.

BUF_GIVEBUF

Give a user-provided buffer to SSM. Refer to the bit-flag in the SPCB used to indicate that user-provided buffers are being used (Ring 3 only). Buffer pointers are passed in *pRetBufTab*.

BUF_EXTENDED

Indicates extended notify structure. It can be used with any other flag.

BUF_GETEMPTY
Get one or more empty buffers to fill.

BUF_RETURNFULL
Return one or more filled buffers.

BUF_RETURNFULL + BUF_RECORDS
Return one or more filled records.

BUF_LASTRECORD
Indicates that last record in a buffer is filled. This can be used with the BUF_RETURNFULL+BUF_RECORDS combination only.

BUF_EOS
Indicates end-of-stream (last buffer produced). This must be used with BUF_RETURNFULL and BUF_GIVEBUF. This flag is passed to the target stream handler with the last buffer in the stream.

BUF_EXTENDEDPTR
Indicates extended notify structures ([EPSRCBUFTAB](#) and [EPTGTBUFTAB](#).) and implies BUF_EXTENDED.

BUF_GDT
Pointers are GDT sel:offset.

Valid Target SH (Consumer) Request Combinations to SMH_NOTIFY:

Any combination of BUF_GETFULL and BUF_RETURNEMPTY can be used for the same SMH_NOTIFY function. As far as the target stream handler is concerned, records are the same as buffers.

Either BUF_LINEAR or BUF_PHYSICAL can be used with any combination of BUF_GETFULL and BUF_RETURNEMPTY. BUF_LINEAR and BUF_PHYSICAL are valid only for device driver stream handlers.

BUF_GETFULL
Get one or more filled buffers to use.

BUF_RETURNEMPTY
Return one or more empty (use) buffers.

BUF_EOS
Received from the SSM with the last buffer in the stream. The target must set this flag when returning the last used buffers.

BUF_EXTENDED
Indicates extended notify structure ([PARM_ENOTIFY](#)). It can be used with any other flag.

BUF_EXTENDEDPTR
Indicates extended notify structures ([EPSRCBUFTAB](#) and [EPTGTBUFTAB](#).) and implies BUF_EXTENDED.

BUF_GDT
Pointers are GDT sel:offset.

SMH_NOTIFY - Example Code

The following code illustrates how to notify the Sync/Stream Manager of a stream buffer request.

```
/*-----  
 * Source Stream Handler Example  
 *-----*/  
  
#include "os2.h"  
#include "os2me.h"  
  
ULONG          ulRC;           /* Error return code          */  
HID            hidSource;      /* Source handler ID          */  
HSTREAM        hstream;       /* Stream handle              */  
PARM_NOTIFY    parm_notify; /* Notify parameter block      */  
SRCBUFTAB      srcbuftab;     /* Source buffer table         */  
PSMHFN         SMHEntryPoint; /* Pointer to SMH entry point */
```

```

/*-----
 * Request an empty stream buffer from the Sync/Stream Manager.
 *-----*/
parm_notify.ulFunction = SMH_NOTIFY; /* Set function */
parm_notify.hid = hidSource; /* Source handler ID */
parm_notify.hstream = hstream; /* Stream handle */

parm_notify.ulFlags = BUF_GETEMPTY; /* Get an empty buffer */
parm_notify.ulGetNumEntries = 1; /* Number of buffers to get */
parm_notify.ulRetNumEntries = 0; /* Not returning any buffers */
parm_notify.pGetBufTab = &srcbuftab; /* Pointer to buffer table */
parm_notify.pRetBufTab = NULL; /* Not returning any buffers */

if (ulRC = SMHEntryPoint (&parm_notify))
    return(ulRC); /* Error! */
.
.
.
/*-----*/
 * Fill the buffer with data.
 *
 * srcbuftab.pBuffer = Pointer to buffer
 * srcbuftab.ulLength = Length of buffer
 *-----*/
.
.
.
/*-----*/
 * Return a full buffer to the Sync/Stream Manager.
 *-----*/
parm_notify.ulFlags = BUF_RETURNFULL; /* Return a full buffer */
parm_notify.ulGetNumEntries = 0; /* Not getting any buffers */
parm_notify.ulRetNumEntries = 1; /* Number of buffers
to return */
parm_notify.pGetBufTab = NULL; /* Not getting any buffers */
parm_notify.pRetBufTab = &srcbuftab; /* Pointer to buffer table */

if (ulRC = SMHEntryPoint (&parm_notify))
    return(ulRC); /* Error! */
/*-----*/
 * Target Stream Handler Example
 *-----*/

#include "os2.h"
#include "os2me.h"

ULONG ulRC; /* Error return code */
HID hidTarget; /* Target handler ID */
HSTREAM hstream; /* Stream handle */
PARAM_NOTIFY parm_notify; /* Notify parameter block */
TGTBUFTAB tgtbuftab; /* Target buffer table */
PSMHFN SMHEntryPoint; /* Pointer to SMH entry point */

/*-----
 * Request a full buffer from the Sync/Stream Manager.
 *-----*/
parm_notify.ulFunction = SMH_NOTIFY; /* Set function */
parm_notify.hid = hidTarget; /* Target handler ID */
parm_notify.hstream = hstream; /* Stream handle */
parm_notify.ulFlags = BUF_GETFULL; /* Get a full buffer */
parm_notify.ulGetNumEntries = 1; /* Number of buffers to get */
parm_notify.ulRetNumEntries = 0; /* Not returning any buffers */
parm_notify.pGetBufTab = &tgtbuftab; /* Pointer to buffer table */
parm_notify.pRetBufTab = NULL; /* Not returning any buffers */

if (ulRC = SMHEntryPoint (&parm_notify))
    return(ulRC); /* Error! */
.
.
.
/*-----
 * Drain the buffer of data.
 *
 * tgtbuftab.pBuffer = Pointer to buffer
 * tgtbuftab.ulLength = Length of buffer
 *-----*/

```

```

        .
        .
        .
*-----
*   Return an empty buffer to the Sync/Stream Manager.
*-----*/
parm_notify.ulFlags = BUF_RETURNEMPTY; /* Return an empty buffer */
parm_notify.ulGetNumEntries = 0;        /* Not getting any buffers */
parm_notify.ulRetNumEntries = 1;        /* Number of buffers to return*/
parm_notify.pGetBufTab = NULL;          /* Not getting any buffers */
parm_notify.pRetBufTab = &tgtbuftab;    /* Pointer to buffer table */

if (ulRC = SMHEntryPoint (&parm_notify))
    return(ulRC);                        /* Error! */

```

SMH_NOTIFY - Topics

Select an item:

- [Description](#)
- [Returns](#)
- [Remarks](#)
- [Example Code](#)
- [Glossary](#)

SMH_REGISTER

SMH_REGISTER Parameter - pParmIn

pParmIn ([PPARM_REG](#))
 A pointer to a [PARM_REG](#) data structure.

SMH_REGISTER Return Value - rc

rc ([ULONG](#))
 Return codes indicating success or the type of failure:

- NO_ERROR**
 Function performed.
- ERROR_INVALID_FUNCTION**
 Invalid function requested.
- ERROR_INVALID_FLAG**
 Invalid input-flag value.

ERROR_INVALID_HNDLR_NAME
Invalid stream-handler name.

ERROR_HNDLR_REGISTERED
Stream handler is already registered.

ERROR_TOO_MANY_HANDLERS
Too many stream handlers have been registered with the Sync/Stream Manager.

ERROR_INVALID_BLOCK
Invalid pointer.

SMH_REGISTER - Description

This message registers a stream handler with the Sync/Stream Manager.

pParmln ([PPARM_REG](#))
A pointer to a [PARM_REG](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR
Function performed.

ERROR_INVALID_FUNCTION
Invalid function requested.

ERROR_INVALID_FLAG
Invalid input-flag value.

ERROR_INVALID_HNDLR_NAME
Invalid stream-handler name.

ERROR_HNDLR_REGISTERED
Stream handler is already registered.

ERROR_TOO_MANY_HANDLERS
Too many stream handlers have been registered with the Sync/Stream Manager.

ERROR_INVALID_BLOCK
Invalid pointer.

SMH_REGISTER - Remarks

Once a DLL stream handler is installed in the system, the stream handler must call the Sync/Stream Manager to register its entry point. After this, the Sync/Stream Manager knows the entry point address for the stream handler, specifically the SHC entry point. Therefore, the stream handler must pass the entry point address to the Sync/Stream Manager on the register call. In addition, when SMH_REGISTER is issued, the stream handler informs the Sync/Stream Manager whether it can be a source or target stream handler. The Sync/Stream Manager, in turn, returns the stream handler IDs for future interaction with the Sync/Stream Manager. The stream handler uses this ID to identify itself whenever calling the Sync/Stream Manager so it will know which stream handler is actually calling it. If a NULL stream handler is being registered, both the source and the target [HID](#) are returned as the same [HID](#). Also, the REGISTER_NONSTREAMING bit flag must be set for NULL stream handlers. The stream-handler name can be a maximum of 8 characters in length to allow device-driver stream handlers to use their device names as the stream-handler name.

SMH_REGISTER - Related Messages

- [SpiGetHandler](#)

SMH_REGISTER - Example Code

The following code illustrates how to register a stream handler with the Sync/Stream Manager.

```
#include      "os2.h"
#include      "os2me.h"

#define      RegName      "C:\PATH\TESTSH" /* Handler name and path */

ULONG      ulRC; /* Error return code */
HID      hidSource, /* Source handler ID */
hidTarget; /* Target handler ID */
PARM_REG      parm_reg; /* Register parameter block */
PSMHFN      SMHEntryPoint; /* Pointer to SMH entry point */

/*-----*/
/* Register a stream handler as both source and target. */
/*-----*/

parm_reg.ulFunction = SMH_REGISTER; /* Set function. */
parm_reg.pszSHName = (PSZ) RegName; /* Set handler name. */
parm_reg.phidSrc = &hidMSrc; /* Returns source hid. */
parm_reg.phidTgt = &hidMTgt; /* Returns target hid. */
parm_reg.ulFlags = REGISTER_TGT_HNDLR | REGISTER_SRC_HNDLR; /* Register as a source
and target stream handler. */

parm_reg.pshcfnEntry = (PSHCFN) main; /* Entry point address. */

if (ulRC = SMHEntryPoint (&parm_reg))
    return(ulRC); /* Error! */
```

SMH_REGISTER - Topics

- Select an item:
- [Description](#)
 - [Returns](#)
 - [Remarks](#)
 - [Related Messages](#)
 - [Example Code](#)
 - [Glossary](#)

SMH_REPORTEVENT

SMH_REPORTEVENT Parameter - pParmIn

pParmIn ([PPARM_EVENT](#))

A pointer to a [PARM_EVENT](#) data structure.

SMH_REPORTEVENT Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Function performed.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_BLOCK

Invalid pointer.

ERROR_INVALID_EVENT

Event type specified is unrecognized.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_STREAM_NOTMASTER

The [HSTREAM](#) specified was not the master stream. This is only possible when reporting sync pulses to the Sync/Stream Manager.

SMH_REPORTEVENT - Description

This message reports events or sync pulses to the Sync/Stream Manager.

pParmIn ([PPARM_EVENT](#))

A pointer to a [PARM_EVENT](#) data structure.

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Function performed.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_BLOCK
Invalid pointer.

ERROR_INVALID_EVENT
Event type specified is unrecognized.

ERROR_INVALID_HID
Invalid handler ID.

ERROR_STREAM_NOTMASTER
The [HSTREAM](#) specified was not the master stream. This is only possible when reporting sync pulses to the Sync/Stream Manager.

SMH_REPORTEVENT - Remarks

A stream handler can use this message to report events, namely, cue points, stream errors, implicit and explicit events, and sync pulses to the Sync/Stream Manager. Stream handlers can detect the following events and report them to the Sync/Stream Manager to be passed back to the application.

- EVENT_CUE_TIME
- EVENT_CUE_TIME_PAUSE
- EVENT_CUE_DATA
- EVENT_ERROR
- EVENT_PLAYLISTMESSAGE (Memory stream handler only)
- EVENT_DATAUNDERRUN
- EVENT_DATAOVERRUN
- EVENT_PLAYLISTCUEPOINT

The Sync/Stream Manager detects the following events and notifies the application directly:

- EVENT_EOS
- EVENT_STREAM_STOPPED
- EVENT_SYNC_PREROLLED
- EVENT_SYNCOVERRUN
- EVENT_QUEUE_OVERFLOW

Each stream handler might or might not be able to generate and receive synchronization pulses. This capability for each stream handler is defined in the SPCBs for that stream handler.

Synchronization pulses are passed as an event from the master stream handler. Synchronization pulses are distributed by the Sync/Stream Manager based on the synchronization relationship of the programmed stream. This distribution is effective for both DLL and device driver slave stream handlers. Device driver stream handlers receive sync pulses by way of their sync pulse SYNC_EVCB. Each slave stream handler must regularly update the sync pulse SYNC_EVCB with its calculated stream time. The Sync/Stream Manager checks this slave-handler stream time against the master stream time and decides whether to send a sync pulse to this handler.

The device driver stream handler checks for sync pulses from the Sync/Stream Manager by polling a flag in the sync-pulse SYNC_EVCB. The Sync/Stream Manager sets the flag to indicate a sync pulse and updates the current master stream time. Typically, the device driver slave stream handler polls the flag once during interrupt processing and adjusts stream usage accordingly.

DLL stream handlers receive sync pulses in one of two ways: either by registering a semaphore with the Sync/Stream Manager, or by the same method as the Sync/Stream Manager uses for device driver stream handler.

SMH_REPORTEVENT - Example Code

The following code illustrates how to report events or sync pulses to the Sync/Stream Manager.

```
#include "os2.h"
#include "os2me.h"

ULONG ulRC; /* Error return code. * /
```

```

HID          hidSource;          /* Source handler ID.          */
TIME_EVCB    timeevcb;          /* Cue point event control block. */
HEVENT       hevent;            /* Time event handle.          */
HSTREAM      hstream;           /* Stream handle.              */
HID          hid;               /* Handler ID.                 */
PARAM_EVENT  parm_event;        /* Report event parameter block. */
MMTIME       mmtimeCurrent;     /* Current stream time.        */
PSMHFN       SMHEntryPoint;     /* Pointer to SMH entry point.  */

/*-----*/
/* Report an event.          */
/*-----*/
parm_event.ulFunction = SMH_REPORTEVENT; /* Set function.          */
parm_event.hid = hidSource;              /* Source handler ID.     */
parm_event.hevent = hevent;              /* Event handle.          */
parm_event.pevcbEvent = (PEVCB) &timeevcb; /* Pointer to Time EVCB. */

timeevcb.ulType = EVENT_CUE_TIME;        /* Set event type.        */
timeevcb.hstream = hstream;              /* Set stream handle.     */
timeevcb.hid = hid;                      /* Set handler ID.        */
timeevcb.ulStatus = 0;                   /* No status.             */
timeevcb.mmtimeStream = mmtimeCurrent;   /* Set current time.      */

if (ulRC = SMHEntryPoint (&parm_event))
    return(ulRC); /* Error! */

```

SMH_REPORTEVENT - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DDCMD Messages

The device driver commands are an interface used by a stream handler to communicate with a physical device driver. This interface uses the inter-device driver communication (IDC) mechanism provided by OS/2 AttachDD DevHelp. The stream handler must attach to the device driver using AttachDD DevHelp to receive the DDCMD entry point address of the device driver. This must be done prior to issuing any DDCMDs. Some device drivers also might require that a DosOpen call be used to open the device driver before it can be used through the AttachDD entry point.

The DDCMDs are provided through a single entry point, [DDCMDEntryPoint](#), which accepts a parameter structure on input.

```

typedef ULONG (FAR*PSHDFN)      (PVOID pParmIn);
typedef ULONG (FAR*PDDCMDFN)    (PVOID pParmIn);

```

For the Ring 3 DLL interface, all pointers are 0:32 linear; for Ring 0 stream handlers all pointers are 16:16 far pointers to enable the 16-bit device driver model to be used. The following list contains the message numbers for all DDCMDs. It should be used in the *ulFunction* field, of the parameter structure passed on the call, to indicate which function is being requested by the stream handler.

Message Number	Message	Description
4L	DDCMD_CONTROL	Performs a device-specific command.
6L	DDCMD_DEREG_STREAM	Removes a stream instance from a device driver.

1L	DDCMD_READ	Performs a read from the device into a buffer.
5L	DDCMD_REG_STREAM	Registers a stream instance with a device driver.
0L	DDCMD_SETUP	Performs a device-specific stream instance setup.
3L	DDCMD_STATUS	Requests status from the device.
2L	DDCMD_WRITE	Performs a write from a buffer to the device.

DDCMDEntryPoint

DDCMDEntryPoint - Syntax

This function enables device driver stream handlers to interface with the hardware physical device driver (PDD).

```
#include <os2.h>

PDDCOMMONCOMMON    pCommon; /* Pointer to DDCMDCOMMON. */
ULONG              rc;      /* Return codes. */

rc = DDCMDEntryPoint(pCommon);
```

DDCMDEntryPoint Parameter - pCommon

pCommon ([PDDCOMMONCOMMON](#)) - input

A pointer to a DDCMD message-specific input parameter block. See [DDCMDCOMMON](#).

DDCMDEntryPoint Return Value - rc

rc ([ULONG](#)) - returns

Return codes indicating success or the type of failure:

NO_ERROR Successful.

ERROR_INVALID_FUNCTION	Invalid function requested.
FAILURE	A DDCMD message-specific error return code.

DDCMDEntryPoint - Parameters

pCommon ([PDDCMDCOMMON](#)) - input
A pointer to a DDCMD message-specific input parameter block. See [DDCMDCOMMON](#).

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Successful.
ERROR_INVALID_FUNCTION	Invalid function requested.
FAILURE	A DDCMD message-specific error return code.

DDCMDEntryPoint - Remarks

Device driver stream handlers communicate with the hardware PDD through the DDCmdEntryPoint. This is accomplished through the use of Device Driver Command (DDCMD) messages, which enable a stream handler to request a PDD to perform functions such as starting, stopping, or resuming operation of a device. This interface uses the IDC mechanism provided by the ATTACHDD DevHelp function. The stream handler must attach to the device driver to receive the DDCMD entry point address of the device driver. This function is performed prior to issuing device driver command messages.

DDCMDEntryPoint - Topics

- Select an item:
- [Syntax](#)
 - [Parameters](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)

DDCMD_CONTROL

DDCMD_CONTROL Parameter - pParmln

pParmln ([PDDCMDCONTROL](#))

A pointer to a [DDCMDCONTROL](#) data structure. The *pParm* and *ulParmSize* fields refer to the [CONTROL_PARM](#) data structure. Values for *ulCmd* include:

DDCMD_START	Start a device.
DDCMD_STOP	Stop a device and return current position.
DDCMD_PAUSE	Pause a device and return current position.
DDCMD_RESUME	Resume a paused device
DDCMD_ENABLE_EVENT	Enable event detection.
DDCMD_DISABLE_EVENT	Disable event detection in the device driver.
DDCMD_PAUSE_TIME	Pause time but do not pause stream.
DDCMD_RESUME_TIME	Resume time.

DDCMD_CONTROL Return Value - rc

rc ([ULONG](#))

Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_REQUEST	Device driver does not support events. (Returned when <i>ulCmd</i> of DDCMDCONTROL is set to DDCMD_ENABLE_EVENT.)
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_SEQUENCE	Invalid device control command.
ERROR_INSUFF_BUFFER	The device driver does not have buffers to perform the requested action.
ERROR_STREAM_NOT_STARTED	The stream cannot perform the requested action unless the stream has been started.
ERROR_TOO_MANY_EVENTS	The stream handler attempted to enable too many events.
FAILURE	Device-driver-specific error return code.

DDCMD_CONTROL - Description

This message performs a device-specific command.

pParmIn (**PDDCMDCONTROL**)

A pointer to a **DDCMDCONTROL** data structure. The *pParm* and *ulParmSize* fields refer to the **CONTROL_PARM** data structure. Values for *ulCmd* include:

- DDCMD_START**
Start a device.
- DDCMD_STOP**
Stop a device and return current position.
- DDCMD_PAUSE**
Pause a device and return current position.
- DDCMD_RESUME**
Resume a paused device
- DDCMD_ENABLE_EVENT**
Enable event detection.
- DDCMD_DISABLE_EVENT**
Disable event detection in the device driver.
- DDCMD_PAUSE_TIME**
Pause time but do not pause stream.
- DDCMD_RESUME_TIME**
Resume time.

rc (**ULONG**)

Error code indicating success or the type of failure:

- NO_ERROR**
Success.
- ERROR_INVALID_FUNCTION**
Illegal function requested.
- ERROR_INVALID_REQUEST**
Device driver does not support events. (Returned when *ulCmd* of **DDCMDCONTROL** is set to **DDCMD_ENABLE_EVENT**.)
- ERROR_INVALID_STREAM**
Invalid stream handle.
- ERROR_INVALID_SEQUENCE**
Invalid device control command.
- ERROR_INSUFF_BUFFER**
The device driver does not have buffers to perform the requested action.
- ERROR_STREAM_NOT_STARTED**
The stream cannot perform the requested action unless the stream has been started.
- ERROR_TOO_MANY_EVENTS**
The stream handler attempted to enable too many events.
- FAILURE**
Device-driver-specific error return code.

DDCMD_CONTROL - Remarks

The stream handler uses this command to control the actions of the physical device driver.

DDCMD_CONTROL - Example Code

The following code illustrates the stream handler requesting the PDD to stop its current task, for example, the PDD stops playing audio.

```
#include      "os2.h"
#include      "os2me.h"
#include      "shdd.h"

ULONG        ulRC;                /* Error return code */
HSTREAM      hstream;             /* Stream handle */
DDCMDCONTROL ddcmdp;              /* Parameter block */
PDDCMDFN     pddcmdfn;            /* Pointer to DDCMD entry point */
.
.
.
/*-----*/
/* The stream handler directs the physical device driver to stop. */
/*-----*/
ddcmdp.ulFunction = DDCMD_CONTROL;
ddcmdp.hstream = hstream;
ddcmdp.pParm = NULL;
ddcmdp.ulParmSize = NULL;
ddcmdp.ulCmd = DDCMD_STOP;

if (ulRC = pddcmdfn (&ddcmdp))
    return (ulRC);    /* error! */
```

DDCMD_CONTROL - Topics

- Select an item:
- [Description](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Glossary](#)
-

DDCMD_DEREG_STREAM

DDCMD_DEREG_STREAM Parameter - pParmIn

pParmln ([PDDCMDDEREGISTER](#))
A pointer to a [DDCMDDEREGISTER](#) data structure.

DDCMD_DEREG_STREAM Return Value - rc

rc ([ULONG](#))
Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
FAILURE	Device-driver-specific error.

DDCMD_DEREG_STREAM - Description

This message removes a stream instance from a device driver. The stream handler directs the device driver to deregister the stream-destroy the stream and all associated data.

pParmln ([PDDCMDDEREGISTER](#))
A pointer to a [DDCMDDEREGISTER](#) data structure.

rc ([ULONG](#))
Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
FAILURE	Device-driver-specific error.

DDCMD_DEREG_STREAM - Remarks

- This message removes the communication link between the physical device driver and the stream handler for a particular stream instance. The only input is the stream handle, which indicates to the device driver which stream to destroy.
- After a deregister, the VSD (or device driver) no longer has access to any buffers, and the buffers will be returned to SSM by the stream handler. The VSD does not have to return the buffers; they are returned for the it. The VSD should not return from the destroy until it has stopped using all buffers.

DDCMD_DEREG_STREAM - Example Code

The following code illustrates the stream handler requesting the stream to be de-registered.

```
#include      "os2.h"
#include      "os2me.h"
#include      "shdd.h"

ULONG        ulRC;                /* Error return code */
HSTREAM      hstream;            /* Stream handle */
DDCMDDEREGISTER ddcmdp;         /* Parameter block */
PDDCMDFN     pddcmdfn;           /* Pointer to DDCMD entry */
                                /* point */
                                .
                                .
                                .
/*-----*/
/*  The stream handler deregisters with the physical device driver. */
/*-----*/
ddcmdpb.ulFunction = DDCMD_DEREG_STREAM;
ddcmdpb.hstream = hstream;

if (ulRC = pddcmdfn (&ddcmdpb))
    return (ulRC);    /* error! */
```

DDCMD_DEREG_STREAM - Topics

Select an item:

- [Description](#)
- [Returns](#)
- [Remarks](#)
- [Example Code](#)
- [Glossary](#)

DDCMD_READ

DDCMD_READ Parameter - pParmIn

pParmln ([PDDCMDREADWRITE](#))
A pointer to a [DDCMDREADWRITE](#) data structure.

DDCMD_READ Return Value - rc

rc ([ULONG](#))
Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_BLOCK	Invalid address passed in parameter block.
FAILURE	Device-driver-specific error return code.

DDCMD_READ - Description

This message performs a read from the device into a buffer.

pParmln ([PDDCMDREADWRITE](#))
A pointer to a [DDCMDREADWRITE](#) data structure.

rc ([ULONG](#))
Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_BLOCK	Invalid address passed in parameter block.
FAILURE	Device-driver-specific error return code.

DDCMD_READ - Remarks

This message is used by a stream handler to give an *empty* buffer to the physical device driver. An example would be in an audio recording, where the physical device driver fills the buffer it gets from the stream handler. The *pBuffer* parameter is a pointer to the buffer to be filled in by the physical device driver. This pointer is either a physical 0:32 pointer, 16:16 far pointer or a global linear pointer. This is defined when the stream handler registers a stream with the device driver using [DDCMD_REG_STREAM](#).

Many devices cannot handle a 0 length buffer. If the driver receives a 0 length buffer, it should not reject the buffer. The driver should do nothing with the buffer and return it in the same order in which it was sent.

DDCMD_READ - Example Code

The following code illustrates the PDD ready to receive an empty buffer from the stream handler.

```
#include      "os2.h"
#include      "os2me.h"
#include      "shdd.h"

ULONG        ulRC;           /* Error return code          */
HSTREAM      hstream;        /* Stream handle              */
DDCMDREADWRITE ddcmdpb;      /* Parameter block            */
PDDCMDDFN    pddcmdfn;       /* Pointer to DDCMD entry point */
PVOID        pBuffer;        /* Pointer to buffer          */

        .
        .
        .

/*-----*/
/* Perform a read from the physical device driver. */
/*-----*/
ddcmdpb.ulFunction = DDCMD_READ;
ddcmdpb.hstream = hstream;
ddcmdpb.pBuffer = pBuffer;
ddcmdpb.ulBufferSize = 32768;

if (ulRC = pddcmdfn (&ddcmdpb))
    return (ulRC); /* error!*/
```

DDCMD_READ - Topics

- Select an item:
- [Description](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Glossary](#)

DDCMD_REG_STREAM

DDCMD_REG_STREAM Parameter - pParmln

pParmln ([PDDCMDREGISTER](#))
A pointer to a [DDCMDREGISTER](#) data structure.

Values for *ulStreamOperation* include:

- STREAM_OPERATION_CONSUME
- STREAM_OPERATION_PRODUCE

Values for *ulAddressType* include:

- ADDRESS_TYPE_VIRTUAL
- ADDRESS_TYPE_PHYSICAL
- ADDRESS_TYPE_LINEAR

DDCMD_REG_STREAM Return Value - rc

rc ([ULONG](#))
Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_HNDLR_REGISTERED	Stream had already been registered with the Sync/Stream Manager.
ERROR_INVALID_SPCBKEY	Invalid SPCBKEY .
ERROR_INITIALIZATION	An error occurred during device initialization.
ERROR_STREAM_CREATION	An error occurred during the creation of this stream instance.
FAILURE	Device-driver-specific error return code.

DDCMD_REG_STREAM - Description

This message registers a stream instance with a device driver for the first time.

pParmln ([PDDCMDREGISTER](#))
A pointer to a [DDCMDREGISTER](#) data structure.

Values for *ulStreamOperation* include:

- STREAM_OPERATION_CONSUME
- STREAM_OPERATION_PRODUCE

Values for *ulAddressType* include:

- ADDRESS_TYPE_VIRTUAL
- ADDRESS_TYPE_PHYSICAL
- ADDRESS_TYPE_LINEAR

rc (ULONG)

Error code indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Illegal function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_HNDLR_REGISTERED

Stream had already been registered with the Sync/Stream Manager.

ERROR_INVALID_SPCBKEY

Invalid SPCBKEY.

ERROR_INITIALIZATION

An error occurred during device initialization.

ERROR_STREAM_CREATION

An error occurred during the creation of this stream instance.

FAILURE

Device-driver-specific error return code.

DDCMD_REG_STREAM - Remarks

A stream handler must register its entry point with the device driver once for each stream instance. This message sets up the communication link between the stream handler and the physical device driver.

DDCMD_REG_STREAM - Example Code

The following code illustrates how to register a stream instance with a device driver.

```
#include "os2.h"
#include "os2me.h"
#include "shdd.h"

ULONG          ulRC;                /* Error return code */
HSTREAM        hstream;            /* Stream handle */
DDCMDREGISTER  ddcmdp;            /* Parameter block */
PDDCMDFN       pddcmdfn;          /* Pointer to DDCMD entry point */

ULONG          ulSysFileNum;        /* Global file handle */
PSHDFN         pshdfn;            /* Pointer to SHD entry point */
SPCBKEY        spcbkey;            /* Stream protocol key */

.
.
.
```

```

/*-----*/
/* Register a stream instance with a physical device driver. */
/*-----*/
ddcmdpb.ulFunction = DDCMD_REGISTER;
ddcmdpb.hstream = hstream;
ddcmdpb.ulSysFileNum = ulSysFileNum;
ddcmdpb.pSHDEntryPoint = pshdfn;
ddcmdpb.ulStreamOperation = STREAM_OPERATION_CONSUME;
ddcmdpb.spcbkey = spcbkey;

if (ulRC = pddcmdfn (&ddcmdpb))
    return (ulRC); /* error!*/

```

DDCMD_REG_STREAM - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DDCMD_SETUP

DDCMD_SETUP Parameter - pParmIn

pParmIn ([PDDCMDSETUP](#))

A pointer to a [DDCMDSETUP](#) data structure. The *pSetupParm* and *ulSetupParmSize* fields refer to the [SETUP_PARM](#) data structure.

DDCMD_SETUP Return Value - rc

rc ([ULONG](#))

Error code indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Illegal function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_REQUEST	Invalid setup request.
ERROR_STREAM_NOT_STOP	The stream cannot perform the requested function unless the stream has been stopped.
FAILURE	Device-driver-specific error return code.

DDCMD_SETUP - Description

This message performs device-specific stream instance setup.

pParmIn ([PDDCMDSETUP](#))

A pointer to a [DDCMDSETUP](#) data structure. The *pSetupParm* and *ulSetupParmSize* fields refer to the [SETUP_PARM](#) data structure.

rc ([ULONG](#))

Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_REQUEST	Invalid setup request.
ERROR_STREAM_NOT_STOP	The stream cannot perform the requested function unless the stream has been stopped.
FAILURE	Device-driver-specific error return code.

DDCMD_SETUP - Remarks

This message indicates to the physical device driver that a specific stream instance will become the active stream instance. The *pSetupParm* field is used for device-specific information. Typically, it is used to pass the current stream time from the stream handler to the PDD because a seek might have been requested on the stream prior to the stream start; thus, the PDD should adjust its time to this new reference time. Time is passed as a pointer to the time in milliseconds.

DDCMD_SETUP - Example Code

The following code illustrates how to perform device-specific stream instance setup.

```
#include "os2.h"
#include "os2me.h"
```



```

#include      "shdd.h"

    ULONG          ulRC;                /* Error return code */
    HSTREAM        hstream;            /* Stream handle */
    DDCMDSETUP     ddcmdpb;            /* Parameter block */
    PDDCMDFN       pddcmdfn;          /* Pointer to DDCMD entry */
                                /* point */
    ULONG          ulStreamTime        /* Stream time */
    PVOID          pBuffer;            /* Pointer to buffer */

                                .
                                .
                                .

/*-----*/
/*Activate a stream instance in a physical device driver (Switch */
/*                               context) */
/*-----*/
ddcmdpb.ulFunction = DDCMD_SETUP;
ddcmdpb.hstream = hstream;
ddcmdpb.pSetupParm = &ulStreamTime; /* Setting stream time */
ddcmdpbp.ulSetupParmSize = sizeof(ulStreamTime);

if (ulRC = pddcmdfn (&ddcmdpb))
    return (ulRC); /* error!*/

```

DDCMD_SETUP - Topics

- Select an item:
- [Description](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Glossary](#)

DDCMD_STATUS

DDCMD_STATUS Parameter - pParmIn

pParmIn ([PDDCMDSTATUS](#))
 A pointer to a [DDCMDSTATUS](#) data structure. The *pStatus* and *ulStatusSize* fields refer to the [STATUS_PARM](#) data structure.

DDCMD_STATUS Return Value - rc

rc ([ULONG](#))
 Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
FAILURE	Device-driver-specific error return code.

DDCMD_STATUS - Description

This message requests status from the device.

pParmln ([PDDCMDSTATUS](#))
A pointer to a [DDCMDSTATUS](#) data structure. The *pStatus* and *ulStatusSize* fields refer to the [STATUS_PARM](#) data structure.

rc ([ULONG](#))
Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
FAILURE	Device-driver-specific error return code.

DDCMD_STATUS - Remarks

This message queries the status of physical device driver. This message might not be supported by all physical device drivers. It is commonly used by the stream handler to request the current stream time from the physical device driver.

DDCMD_STATUS - Example Code

The following code illustrates how to request status from the device.

```
#include "os2.h"
#include "os2me.h"
#include "shdd.h"

ULONG          ulRC;           /* Error return code */
HSTREAM        hstream;       /* Stream handle */
DDCMDSTATUS    ddcmdp;       /* Parameter block */
PDDCMDFN       pddcmdfn;      /* Pointer to DDCMD entry point
```

```

        .
        .
        .
/*-----*/
/*  Get the current stream time from the physical device driver.  */
/*-----*/
ddcmdpb.ulFunction = DDCMD_STATUS;
ddcmdpb.hstream = hstream;
ddcmdpb.pStatus = NULL;                /* Return stream time*/
ddcmdpb.ulStatusSize = 0;

if (ulRC = pddcmdfn (&ddcmdpb))
    return (ulRC);    /* error!*/

```

DDCMD_STATUS - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

DDCMD_WRITE

DDCMD_WRITE Parameter - pParmln

pParmln ([PDDCMDREADWRITE](#))

A pointer to a [DDCMDREADWRITE](#) data structure.

DDCMD_WRITE Return Value - rc

rc ([ULONG](#))

Error code indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Illegal function requested.

ERROR_INVALID_STREAM

	Invalid stream handle.
ERROR_INVALID_BLOCK	Invalid address passed in parameter block.
FAILURE	Device-driver-specific error return code.

DDCMD_WRITE - Description

This message performs a write from the buffer to the device.

pParmln ([PDDCMDREADWRITE](#))

A pointer to a [DDCMDREADWRITE](#) data structure.

rc ([ULONG](#))

Error code indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_BLOCK	Invalid address passed in parameter block.
FAILURE	Device-driver-specific error return code.

DDCMD_WRITE - Remarks

This message is used by a stream handler to give a full buffer to the physical device driver. An example would be the case of audio playback, where the stream handler passes a buffer with data to the physical device driver.

The *pBuffer* parameter is a pointer to the data buffer. Note that this pointer is either a physical 0:32 pointer, 16:16 far pointer, or a global linear pointer. The pointer type is defined when the stream registers a stream with the device driver ([DDCMD_REG_STREAM](#)).

Many devices cannot handle a 0 length buffer. If the driver receives a 0 length buffer, it should not reject the buffer. The driver should do nothing with the buffer and return it in the same order in which it was sent.

DDCMD_WRITE - Example Code

The following code illustrates how to perform a write to the physical device driver.

```
#include "os2.h"
#include "os2me.h"
#include "shdd.h"
```

```

ULONG          ulRC;                /* Error return code */
HSTREAM        hstream;             /* Stream handle */
DDCMDREADWRITE ddcmdp;             /* Parameter block */
PDDCMDFN       pddcmdfn;            /* Pointer to DDCMD entry */
/* point */
PVOID          pBuffer;             /* Pointer to buffer */

.
.
.
/*-----*/
/* Perform a write to the physical device driver. */
/*-----*/
ddcmdpb.ulFunction = DDCMD_WRITE;
ddcmdpb.hstream = hstream;
ddcmdpb.pBuffer = pBuffer;
ddcmdpb.ulBufferSize = 32768;

if (ulRC = pddcmdfn (&ddcmdpb))
    return (ulRC); /* error! */

```

DDCMD_WRITE - Topics

Select an item:

- [Description](#)
- [Returns](#)
- [Remarks](#)
- [Example Code](#)
- [Glossary](#)

VSD Commands

The vendor-specific driver (VSD) interface is a generic device-driver like interface that effectively decomposes the high-level media control interface functions into more fundamental operations. The VSD layer resides in Ring 3 and provides a low-level function set that can keep pace with demanding data control and transport requirements. A Ring 3 stream handler can use the [VSD_DDCMD](#) command (described in this section) for data transportation. For this Ring 3 DLL interface, all pointers are 0:32 linear.

Note: Refer to the *MMPM/2 Device Driver Reference* for a complete description of the VSD interface definitions.

The interface to the VSD is a specific entry point-[VSDEntry](#). First, issue DosLoadModule for the VSD's DLL. The DosLoadModule function returns a handle for the VSD. Next, using the VSD handle, call DosQueryProcAddr to receive the VSD entry point address. This must be done prior to issuing any VSD commands. Once the entry point address is received, calls to the VSD can be made with [VSDEntry](#).

VSDEntry

VSDEntry - Syntax

This entry point enables communication between vendor-specific drivers (VSDs) and an application such as the user-level audio stream handler or amplifier mixer. For audio VSDs, be sure to define INCL_AUDIO_VSD before including the VSDCMDS.H header file.

```
#define INCL_AUDIO_VSD
#include <vsdcmds.h>
#include <os2mixer.h>

HVSD    hvsd;      /* Handle to VSD instance. */
ULONG   ulFunc;    /* Function code. */
ULONG   ulFlags;   /* Flags for driver. */
PVOID   pRequest;  /* Request parameter block value. */
ULONG   rc;        /* Return codes. */

rc = VSDEntry(hvsd, ulFunc, ulFlags, pRequest);
```

VSDEntry Parameter - hvsd

hvsd (HVSD) - input
This parameter is the handle to the VSD instance.

VSDEntry Parameter - ulFunc

ulFunc (ULONG) - input
The VSD command to be issued. The following commands are supported for audio VSDs.

Command	Description
VSD_CLOSE	Closes the device. (Required)
VSD_DDCMD	Allows communication between stream handlers and their attached devices. (Required)
VSD_ESCAPE	Sends a buffer to the device. (Optional)
VSD_GETDEVCAPS	Retrieves the device capabilities. (Required)
VSD_OPEN	Opens an instance of the device. (Required)
VSD_QUERY	Queries the status of the device. (Required)
VSD_RESOURCE	Manages resources. (Required)
VSD_RESTORE	Restores device to a saved state. (Required)
VSD_SAVE	Saves the current state of the device instance. (Required)
VSD_SET	Modifies settings of the device. (Required)

VSD_USER	Allows user-defined commands to be passed into the VSD. (Optional)
----------	--

VSDEntry Parameter - ulFlags

ulFlags (ULONG) - input

The *ulFlags* parameter is used to further qualify the command specified in *ulFunc*. In many cases it is used as a subcommand. For more information on *ulFlags*, see the specific *ulFunc* parameter.

VSDEntry Parameter - pRequest

pRequest (PVOID) - input

Specifies the pointer to the request packet. The caller of the VSD supplies all request buffers. See individual commands for more detailed information.

VSDEntry Return Value - rc

rc (ULONG) - returns

The return codes are listed for each command and will vary based on the command sent.

If a VSD requires unique return codes for certain conditions, it can use return codes starting at VSDERR_BASE.

VSDEntry - Parameters

hvsd (HVSD) - input

This parameter is the handle to the VSD instance.

ulFunc (ULONG) - input

The VSD command to be issued. The following commands are supported for audio VSDs.

Command	Description
VSD_CLOSE	Closes the device. (Required)
VSD_DDCMD	Allows communication between stream handlers and their attached devices. (Required)
VSD_ESCAPE	Sends a buffer to the device. (Optional)

VSD_GETDEVCAPS	Retrieves the device capabilities. (Required)
VSD_OPEN	Opens an instance of the device. (Required)
VSD_QUERY	Queries the status of the device. (Required)
VSD_RESOURCE	Manages resources. (Required)
VSD_RESTORE	Restores device to a saved state. (Required)
VSD_SAVE	Saves the current state of the device instance. (Required)
VSD_SET	Modifies settings of the device. (Required)
VSD_USER	Allows user-defined commands to be passed into the VSD. (Optional)

ulFlags ([ULONG](#)) - input

The *ulFlags* parameter is used to further qualify the command specified in *ulFunc*. In many cases it is used as a subcommand. For more information on *ulFlags*, see the specific *ulFunc* parameter.

pRequest ([PVOID](#)) - input

Specifies the pointer to the request packet. The caller of the VSD supplies all request buffers. See individual commands for more detailed information.

rc ([ULONG](#)) - returns

The return codes are listed for each command and will vary based on the command sent.

If a VSD requires unique return codes for certain conditions, it can use return codes starting at VSDERR_BASE.

VSDEntry - Remarks

If a VSD requires unique return codes for certain conditions, it can use return codes starting at VSDERR_BASE.

VSDEntry - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Glossary](#)

VSD_DDCMD

VSD_DDCMD Parameter - hvsd

hvsd ([HVSD](#))

Handle to a VSD instance.

VSD_DDCMD Parameter - ulFunc

ulFunc ([ULONG](#))

Set to VSD_DDCMD.

VSD_DDCMD Parameter - ulFlags

ulFlags ([ULONG](#))

The following flags (subcommands) are defined for this command:

DDCMD_CONTROL

This subcommand of VSD_DDCMD performs a device-specific command. *pRequest* is a pointer to the [DDCMDCONTROL](#) structure.

DDCMD_DEREG_STREAM

This subcommand of VSD_DDCMD deregisters a stream instance with the device driver. *pRequest* is a pointer to [DDCMDDEREGISTER](#) structure.

DDCMD_READ

This subcommand of VSD_DDCMD is used by a stream handler to give an *empty* buffer to the VSD (for example, during a record operation). When the buffer has been filled, the VSD is responsible for communicating to the caller that the buffer has been filled. The VSD should pass a [MSG_REPORTINT](#) structure to the *pSHDEntryPoint* in [DDCMDREGISTER](#) to inform the caller.

pRequest is a pointer to [DDCMDREADWRITE](#) structure.

DDCMD_REG_STREAM

This subcommand of VSD_DDCMD registers a stream instance with the device driver. *pRequest* is a pointer to [DDCMDREGISTER](#) structure.

DDCMD_SETUP

This subcommand of VSD_DDCMD performs device-specific stream instance setup. *pRequest* is a pointer to [DDCMDSETUP](#) structure.

DDCMD_STATUS

This VSD subcommand requests streaming status from a device. Typically, it is called to request the current stream time. *pRequest* is a pointer to [DDCMDSTATUS](#) structure.

DDCMD_WRITE

This VSD subcommand is used by a stream handler to give a *full* buffer to the VSD (for example, during a playback operation). When the buffer has been consumed, the VSD is responsible for communicating to the caller that the buffer has been used. The VSD should pass a [MSG_REPORTINT](#) structure to the *pSHDEntryPoint* in [DDCMDREGISTER](#) to inform the caller.

pRequest is a pointer to [DDCMDREADWRITE](#) structure.

VSD_DDCMD Parameter - pRequest

pRequest (PVOID)

The value of *pRequest* varies according to the *ulFlags* value. See each particular *ulFlags* value for the definition of *pRequest*.

VSD_DDCMD Return Value - rc

rc (ULONG)

Possible error codes vary according to the value of *ulFlags*.

VSD_DDCMD - Description

Stream handlers can communicate with their attached devices using the VSD_DDCMD calls. The DDCMD interface is the primary method of moving data to and from devices in OS/2 multimedia.

This command is sent using [VSDEntry](#) as follows:

```
VSDEntry(hvsd, ulFunc, ulFlags, pRequest)
```

hvsd (HVSD)

Handle to a VSD instance.

ulFunc (ULONG)

Set to VSD_DDCMD.

ulFlags (ULONG)

The following flags (subcommands) are defined for this command:

DDCMD_CONTROL

This subcommand of VSD_DDCMD performs a device-specific command. *pRequest* is a pointer to the [DDCMDCONTROL](#) structure.

DDCMD_DEREG_STREAM

This subcommand of VSD_DDCMD deregisters a stream instance with the device driver. *pRequest* is a pointer to [DDCMDDEREGISTER](#) structure.

DDCMD_READ

This subcommand of VSD_DDCMD is used by a stream handler to give an *empty* buffer to the VSD (for example, during a record operation). When the buffer has been filled, the VSD is responsible for communicating to the caller that the buffer has been filled. The VSD should pass a [MSG_REPORTINT](#) structure to the *pSHDEntryPoint* in [DDCMDREGISTER](#) to inform the caller.

pRequest is a pointer to [DDCMDREADWRITE](#) structure.

DDCMD_REG_STREAM

This subcommand of VSD_DDCMD registers a stream instance with the device driver. *pRequest* is a pointer to [DDCMDREGISTER](#) structure.

DDCMD_SETUP
This subcommand of VSD_DDCMD performs device-specific stream instance setup. *pRequest* is a pointer to [DDCMDSETUP](#) structure.

DDCMD_STATUS
This VSD subcommand requests streaming status from a device. Typically, it is called to request the current stream time. *pRequest* is a pointer to [DDCMDSTATUS](#) structure.

DDCMD_WRITE
This VSD subcommand is used by a stream handler to give a *full* buffer to the VSD (for example, during a playback operation). When the buffer has been consumed, the VSD is responsible for communicating to the caller that the buffer has been used. The VSD should pass a [MSG_REPORTINT](#) structure to the *pSHDEntryPoint* in [DDCMDREGISTER](#) to inform the caller.

pRequest is a pointer to [DDCMDREADWRITE](#) structure.

pRequest (PVOID)
The value of *pRequest* varies according to the *ulFlags* value. See each particular *ulFlags* value for the definition of *pRequest*.

rc (ULONG)
Possible error codes vary according to the value of *ulFlags*.

VSD_DDCMD - Topics

Select an item:
[Description](#)
[Returns](#)
[Glossary](#)

SHD Messages

The stream handler device (SHD) messages are provided by a stream handler through a single entry point, [SHDEntryPoint](#), to a physical device driver. For this interface, all pointers are 16:16 pointers; this enables the current 16-bit device-driver model to be used for stream handlers. There are two SHD messages. The message number must be used in the *ulFunction* field, of the parameter structure passed in the call, to indicate which message is being requested by the stream handler.

The following table lists the SHD messages:

Message Number	Message	Description
1L	SHD_REPORT_EVENT	Reports an event to the stream handler.
0L	SHD_REPORT_INT	Reports an interrupt from a device.

SHDEntryPoint

SHDEntryPoint - Syntax

Each stream handler must provide this entry point, which is used for the PDD to communicate back to the stream handler.

```
#include <os2.h>

PSHD_COMMON    pCommon; /* Pointer to SHD_COMMON. */
ULONG          rc;      /* Return codes. */

rc = SHDEntryPoint(pCommon);
```

SHDEntryPoint Parameter - pCommon

pCommon ([PSHD_COMMON](#)) - input
Pointer to [SHD_COMMON](#).

SHDEntryPoint Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Successful.
ERROR_INVALID_FUNCTION	Invalid function requested.
FAILURE	An SHD message-specific error return code.

SHDEntryPoint - Parameters

pCommon ([PSHD_COMMON](#)) - input
Pointer to [SHD_COMMON](#).

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Successful.
ERROR_INVALID_FUNCTION	Invalid function requested.
FAILURE	An SHD message-specific error return code.

SHDEntryPoint - Remarks

Device driver stream handlers receive commands from PDDs to report events and interrupts. These Stream Handler Device (SHD) helper commands are provided through the SHDEntryPoint. This entry point is specifically used for the PDD to call back to the stream handler. For example, the PDD can send an [SHD_REPORT_INT](#) command to the stream handler to report status, indicate that a buffer is full, or specify that an additional buffer is required.

SHDEntryPoint - Example Code

The following code illustrates how to access this entry point, which is used for the PDD to communicate back to the stream handler.

```
#include "os2.h"
#include "os2me.h"
#include "shdd.h"

ULONG          ulRC;                /* Error return code */
HSTREAM        hstream;             /* Stream handle */
HEVENT         hevent;              /* Event handle */
SHD_REPORTEVENT shdpb;              /* Parameter block */
PSHDFN         pshdfn;              /* Pointer to SHD entry point */
ULONG          ulStreamTime;        /* Stream time */

        .
        .
        .

/*-----*/
/* Report a cue point to the stream handler for a stream instance. */
/*-----*/
shdpb.ulFunction = SHD_REPORT_EVENT;
shdpb.hstream = hstream;
shdpb.hevent = hevent;
shdpb.ulStreamTime = ulStreamTime;

if (ulRC = pshdfn (&shdpb))
    return (ulRC); /* error! */
```

SHDEntryPoint - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Glossary](#)

SHD_REPORT_EVENT

SHD_REPORT_EVENT Parameter - pParmln

pParmln ([PSHD_REPORTEVENT](#))
A pointer to an [SHD_REPORTEVENT](#) data structure.

SHD_REPORT_EVENT Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_EVENT	Invalid event handle.
FAILURE	Stream-handler-specific error return code.

SHD_REPORT_EVENT - Description

This message reports an event to the stream handler.

pParmln ([PSHD_REPORTEVENT](#))
A pointer to an [SHD_REPORTEVENT](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Illegal function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_EVENT	Invalid event handle.
FAILURE	Stream-handler-specific error return code.

SHD_REPORT_EVENT - Remarks

This message is a mechanism for the physical device driver to report its own event (cue point) detection. The stream handler will call the PDD with an event via [DDCMD_CONTROL](#) (DDCMD_ENABLE_EVENT) and the PDD will monitor the stream time for this event (if the PDD supports event detection). When the event is detected, the PDD will call back the stream handler and report this event via this message. The handle to the event and stream time must be set.

SHD_REPORT_EVENT - Example Code

The following code illustrates how to report an event to the stream handler.

```
#include      "os2.h"
#include      "os2me.h"
#include      "shdd.h"

ULONG        ulRC;                /* Error return code */
HSTREAM      hstream;             /* Stream handle */
HEVENT       hevent;              /* Event handle */
SHD_REPORTEVENT shdpb;            /* Parameter block */
PSHDFN       pshdfn;              /* Pointer to SHD entry point */
ULONG        ulStreamTime;        /* Stream time */

.
.
.

/*-----*/
/*  Report a cue point to the stream handler for a stream instance.  */
/*-----*/
shdpb.ulFunction = SHD_REPORT_EVENT;
shdpb.hstream = hstream;
shdpb.hevent = hevent;
shdpb.ulStreamTime = ulStreamTime;

if (ulRC = pshdfn (&shdpb))
    return (ulRC);    /* error! */
```

SHD_REPORT_EVENT - Topics

- Select an item:
- [Description](#)
 - [Returns](#)
 - [Remarks](#)
 - [Example Code](#)
 - [Glossary](#)
-

SHD_REPORT_INT

SHD_REPORT_INT Parameter - pParmln

pParmln ([PSHD_REPORTINT](#))

A pointer to a [SHD_REPORTINT](#) data structure.

SHD_REPORT_INT Return Value - rc

rc ([ULONG](#))

Return codes indicating success or type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Illegal function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_DEVICE_UNDERRUN

There was a device data underrun.

ERROR_DEVICE_OVERRUN

There was a device data overrun.

FAILURE

Stream-handler-specific error return code.

SHD_REPORT_INT - Description

This message is used by the physical device driver to report interrupts from a device and indicate that a new buffer is needed for consumption.

pParmln ([PSHD_REPORTINT](#))

A pointer to a [SHD_REPORTINT](#) data structure.

rc ([ULONG](#))

Return codes indicating success or type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Illegal function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_DEVICE_UNDERRUN

There was a device data underrun.

ERROR_DEVICE_OVERRUN
There was a device data overrun.

FAILURE
Stream-handler-specific error return code.

SHD_REPORT_INT - Remarks

This message is a mechanism for the physical device driver to return status, indicate that the buffer has been consumed, or indicate that a new buffer is needed.

When status is returned, you will know whether the playback or record was successful or if there was an error condition. Error conditions for playback are known as *underruns*, which means the device is not getting data fast enough. Error conditions for records are called *overruns*, which means the device is generating data faster than empty buffers become available. This might result in a data loss situation. For example, the device is constantly bringing in data from the analog ports of the adapter and converting it to digital data. If the device does not have any empty buffers to store the digital data, the data is lost.

If you report an underrun under Warp and want the stream handler to pause the device, you should report the ERROR_DEVICE_UNDERRUN as well as the SHD_WRITE_COMPLETE flags.

The VSD must return buffers in the order they were sent and should not hold on to any buffers. Failure to return buffers could result in application hangs.

SHD_REPORT_INT - Example Code

The following code illustrates how to report an interrupt from a device.

```
#include      "os2.h"
#include      "os2me.h"
#include      "shdd.h"

ULONG        ulRC;                /* Error return code */
HSTREAM      hstream;            /* Stream handle */
SHD_REPORTINT shdpb;             /* Parameter block */
PSHDFN       pshdfn;            /* Pointer to SHD entry point */
PVOID        pBuffer;           /* Pointer to buffer */
ULONG        ulStreamTime;       /* Stream time in millisecs */

.
.
.

/*-----*/
/* Report a read has completed. */
/*-----*/
shdpb.ulFunction = SHD_REPORT_INT;
shdpb.hstream = hstream;
shdpb.pBuffer = pBuffer;
shdpb.ulFlag = SHD_READ_COMPLETE;
shdpb.ulStatus = LengthRecordedBuffer;
shdpb.ulStreamTime = ulStreamTime;

if (ulRC = pshdfn (&shdpb))
    return (ulRC);    /* error! */
```

SHD_REPORT_INT - Topics

Select an item:

- Description
- Returns
- Remarks
- Example Code
- Glossary

SHC Messages

Stream handler command messages are provided by each stream handler in the Sync/Stream subsystem. Stream handlers can be DLLs or device drivers. All SHC messages are synchronous and must be provided by both DLL stream handlers (as a DLL call) and by device driver stream handlers as an inter-device driver communication (IDC) call. SHC messages are issued only from the Sync/Stream Manager and are not used directly by an application media control device.

The stream handler commands are provided through a single entry point, [SHCEntryPoint](#), which accepts a parameter structure as input. This permits the DLL and the device driver interfaces to the Sync/Stream Manager to be the same. The entry point for DLL stream handlers is registered with the Sync/Stream Manager DLL during the stream handler DLL initialization routine called by the loader. The entry point for the device driver stream handlers is registered with the Sync/Stream Manager device driver during device driver initialization. For the DLL interface, all pointers are 0:32 linear. For the device driver interface, pointers are 16:16 or 0:32 physical or global linear. This enables the current 16-bit device driver model to be used for stream handlers.

The following table lists the message numbers for all stream handler commands (SHCs) that must be supported by all stream handlers. These are used in the *uiFunction* field, of the parameter structure passed with the call, to indicate which message is being requested by the Sync/Stream Manager.

Message Number	Message	Description
0L	SHC_ASSOCIATE	Associates a data object with a stream handler.
1L	SHC_CLOSE	Closes a stream handler.
2L	SHC_CREATE	Creates a stream instance for a stream handler.
3L	SHC_DESTROY	Removes a stream instance for a stream handler.
8L	SHC_DISABLE_EVENT	Disables event notification for a particular event.
10L	SHC_DISABLE_SYNC	Disables synchronization for a stream handler in a sync group.
7L	SHC_ENABLE_EVENT	Enables event notification for a particular event.
9L	SHC_ENABLE_SYNC	Enables synchronization for a stream handler in a sync group.
14L	SHC_ENUMERATE_PROTOCOLS	Returns a list of stream protocol keys for the specified stream handler.
12L	SHC_GET_PROTOCOL	Queries a stream handler for a specified stream protocol.
11L	SHC_GET_TIME	Queries the current stream time.
13L	SHC_INSTALL_PROTOCOL	Installs or removes a specified stream protocol

		for a stream handler.
15L	SHC_NEGOTIATE_RESULT	Provides the results of a SPCB negotiation for a stream instance.
6L	SHC_SEEK	Seeks to a specified point in the stream object or sets the current stream time.
16L	SHC_SENDMSG	Sends specific message to stream handler.
4L	SHC_START	Starts data streaming for a stream handler of a particular stream instance.
5L	SHC_STOP	Stops data streaming for a stream handler of a particular stream instance.

SHCEntryPoint

SHCEntryPoint - Syntax

Each stream handler must provide this entry point, which is used by the Sync/Stream Manager to communicate with the stream handler.

```
#include <os2.h>

PVOID ParmIn; /* Pointer to SHC message-specific structure. */
ULONG rc;     /* Return codes. */

rc = SHCEntryPoint(ParmIn);
```

SHCEntryPoint Parameter - ParmIn

ParmIn ([PVOID](#)) - input
 Pointer to an SHC message-specific input parameter block. See [SHC_COMMON](#).

SHCEntryPoint Return Value - rc

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Successful.
ERROR_INVALID_FUNCTION	Invalid function requested.
FAILURE	An SHC message-specific error return code.

SHCEntryPoint - Parameters

Parmln ([PVOID](#)) - input
Pointer to an SHC message-specific input parameter block. See [SHC_COMMON](#).

rc ([ULONG](#)) - returns
Return codes indicating success or the type of failure:

NO_ERROR	Successful.
ERROR_INVALID_FUNCTION	Invalid function requested.
FAILURE	An SHC message-specific error return code.

SHCEntryPoint - Remarks

DLL stream handlers must supply an interface to communicate with the Sync/Stream Manager. This interface, called the stream handler command (SHC) interface, exports stream handler commands (SHCs) for use by the Sync/Stream Manager to set up and control data streams. These SHCs are accessible through a single entry point, SHCEntryPoint.

In addition, device driver stream handlers receive commands from the Sync/Stream Manager to initialize and perform streaming functions. These stream handler commands (SHCs) are also accessible through SHCEntryPoint. The main routine is an IDC interface with the Sync/Stream Manager Device Driver. The SSM calls the device driver stream handler by issuing stream programming interface (SPI) functions such as [SpiCreateStream](#), [SpiStartStream](#), and [SpiStopStream](#).

SHCEntryPoint - Topics

Select an item:

- [Syntax](#)
- [Parameters](#)
- [Returns](#)
- [Remarks](#)
- [Glossary](#)

SHC_ASSOCIATE

SHC_ASSOCIATE Parameter - pParmIn

pParmIn ([PPARM_ASSOC](#))

A pointer to a [PARM_ASSOC](#) data structure.

SHC_ASSOCIATE Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_REQUEST

The stream handler does not support the associate function.

ERROR_ACCESS_OBJECT

Cannot access the specified data object.

ERROR_INVALID_OBJTYPE

The object type specified is unknown.

ERROR_STREAM_NOT_STOP

The stream cannot perform the requested function unless the stream has been stopped.

FAILURE

Stream handler-specific error return code.

SHC_ASSOCIATE - Description

This message associates a data object with a stream handler.

pParmIn ([PPARM_ASSOC](#))

A pointer to a [PARM_ASSOC](#) data structure.

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_REQUEST	The stream handler does not support the associate function.
ERROR_ACCESS_OBJECT	Cannot access the specified data object.
ERROR_INVALID_OBJTYPE	The object type specified is unknown.
ERROR_STREAM_NOT_STOP	The stream cannot perform the requested function unless the stream has been stopped.
FAILURE	Stream handler-specific error return code.

SHC_ASSOCIATE - Remarks

Object handles are interpreted on the basis of the stream data type. The stream time must not be reset to 0 after a new object is associated. The stream can be searched back to 0 using [SHC_SEEK](#). This permits SHC_ASSOCIATE to be used for more than associating data objects with a stream; it also can be used to pass information to the stream handler.

SHC_ASSOCIATE - Related Messages

- [SpiAssociate](#)

SHC_ASSOCIATE - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

SHC_CLOSE

SHC_CLOSE Parameter - pParmln

pParmln ([PPARM_CLOSE](#))
A pointer to a [PARM_CLOSE](#) data structure.

SHC_CLOSE Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_HID	Invalid handler ID.
FAILURE	Stream-handler-specific error return code.

SHC_CLOSE - Description

This message closes a stream handler.

pParmln ([PPARM_CLOSE](#))
A pointer to a [PARM_CLOSE](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_HID	Invalid handler ID.
FAILURE	Stream-handler-specific error return code.

SHC_CLOSE - Remarks

A call to terminate the use of the stream handler by the Sync/Stream Manager is made when all processes using the SPI functions have terminated. Only device driver stream handlers are called with SHC_CLOSE to clean up any resources still allocated, including any stream protocols (SPCBs) that are installed in the stream handler. DLL stream handlers are notified of termination through a DLL termination routine.

SHC_CLOSE - Related Messages

- [SpiGetHandler](#)
- [SpiInstallProtocol](#)

SHC_CLOSE - Topics

Select an item:
[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

SHC_CREATE

SHC_CREATE Parameter - pParmIn

pParmIn ([PPARM_CREATE](#))
A pointer to a [PARM_CREATE](#) data structure.

SHC_CREATE Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_PARAMETER	Invalid parameter.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_SPCBKEY	Invalid SPCBKEY .
ERROR_DEVICE_NOT_FOUND	Invalid device driver name.
ERROR_STREAM_CREATION	Error creating the stream..
ERROR_INVALID_BLOCK	Invalid DCB pointer.
ERROR_ALLOC_RESOURCES	Error allocating a system resource. Out of memory or similar error.
FAILURE	Stream-handler-specific error return code.

SHC_CREATE - Description

This message creates a stream instance for a stream handler.

pParmIn ([PPARM_CREATE](#))

A pointer to a [PARM_CREATE](#) data structure.

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_PARAMETER	Invalid parameter.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_SPCBKEY	Invalid SPCBKEY .
ERROR_DEVICE_NOT_FOUND	

	Invalid device driver name.
ERROR_STREAM_CREATION	Error creating the stream..
ERROR_INVALID_BLOCK	Invalid DCB pointer.
ERROR_ALLOC_RESOURCES	Error allocating a system resource. Out of memory or similar error.
FAILURE	Stream-handler-specific error return code.

SHC_CREATE - Remarks

This message creates a stream instance with *hstream* as the handle. The Sync/Stream Manager calls both the source and target stream handlers during an [SpiCreateStream](#) call from the application or media driver. The stream handler must pass back a pointer to an [SPCB](#) so that the Sync/Stream Manager can negotiate the stream parameters. The Sync/Stream Manager calls [SHC_NEGOTIATE_RESULT](#) with the results of the stream negotiation. Device driver stream handlers must perform the OS/2 function AttachDD (DevHelp service), based on the device information in the device control block.

Stream handlers must be able to handle multiple stream instances. These stream instances are created when an SHC_CREATE message is received from the Sync/Stream Manager. The stream handler should create any stream instance control structures at this point.

SHC_CREATE - Related Messages

- [SpiCreateStream](#)

SHC_CREATE - Related Messages

- [SHC_NEGOTIATE_RESULT](#)

SHC_CREATE - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Related Messages](#)
[Glossary](#)

SHC_DESTROY

SHC_DESTROY Parameter - pParmln

pParmln ([PPARM_DESTROY](#))
A pointer to a [PARM_DESTROY](#) data structure.

SHC_DESTROY Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_HID	Invalid handler ID.
FAILURE	Stream handler-specific error return code.

SHC_DESTROY - Description

This message removes a stream instance for a stream handler.

pParmln ([PPARM_DESTROY](#))
A pointer to a [PARM_DESTROY](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_HID	

	Invalid handler ID.
FAILURE	Stream handler-specific error return code.

SHC_DESTROY - Remarks

This message does the appropriate cleanup of stream-specific resources for this stream instance. If this is a master stream in a synchronized stream relationship, the slave streams are sent [SHC_DISABLE_SYNC](#).

SHC_DESTROY - Related Messages

- [SpiDestroyStream](#)

SHC_DESTROY - Topics

Select an item:

- [Description](#)
- [Returns](#)
- [Remarks](#)
- [Related Messages](#)
- [Glossary](#)

SHC_DISABLE_EVENT

SHC_DISABLE_EVENT Parameter - pParmIn

pParmIn ([PPARM_DISEVENT](#))
A pointer to a [PARM_DISEVENT](#) data structure.

SHC_DISABLE_EVENT Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_EVENT	Invalid event handle.
ERROR_INVALID_EVCB	Invalid event control block.
FAILURE	Stream handler-specific error return code.

SHC_DISABLE_EVENT - Description

This message disables event notification for a particular event.

pParmIn ([PPARM_DISEVENT](#))
A pointer to a [PARM_DISEVENT](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_EVENT	Invalid event handle.
ERROR_INVALID_EVCB	Invalid event control block.
FAILURE	Stream handler-specific error return code.

SHC_DISABLE_EVENT - Remarks

Commands the stream handler to disable generation of a specified explicit event. Only explicit events can be enabled or disabled. The stream handler must no longer report events of this type to the Sync/Stream Manager.

SHC_DISABLE_EVENT - Related Messages

- [SpiDisableEvent](#)

SHC_DISABLE_EVENT - Topics

Select an item:
[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

SHC_DISABLE_SYNC

SHC_DISABLE_SYNC Parameter - pParmIn

pParmIn ([PPARM DISSYNC](#))
A pointer to a [PARM DISSYNC](#) data structure.

SHC_DISABLE_SYNC Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

<code>NO_ERROR</code>	Success.
<code>ERROR_INVALID_FUNCTION</code>	Invalid function requested.
<code>ERROR_INVALID_STREAM</code>	Invalid stream handle.

ERROR_INVALID_HID	Invalid handler ID.
FAILURE	Stream-handler-specific error return code.

SHC_DISABLE_SYNC - Description

This message disables synchronization for a stream handler in a sync group.

pParmIn ([PPARM_DISSYNC](#))
A pointer to a [PARM_DISSYNC](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_HID	Invalid handler ID.
FAILURE	Stream-handler-specific error return code.

SHC_DISABLE_SYNC - Remarks

Commands the stream handler to disable synchronization for this stream instance.

SHC_DISABLE_SYNC - Related Messages

- [SpiDisableSync](#)

SHC_DISABLE_SYNC - Topics

Select an item:
[Description](#)
[Returns](#)

SHC_ENABLE_EVENT

SHC_ENABLE_EVENT Parameter - pParmln

pParmln ([PPARM_ENEVEN](#)T)

A pointer to a [PARM_ENEVEN](#)T data structure.

SHC_ENABLE_EVENT Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_EVENT

Invalid event handle.

ERROR_INVALID_HID

Invalid handle ID.

ERROR_INVALID_EVCB

Invalid event control block.

ERROR_TOO_MANY_EVENTS

Tried to enable too many events.

FAILURE

Stream-handler-specific error return code.

SHC_ENABLE_EVENT - Description

This message enables event notification for a particular event.

pParmIn ([PPARM_ENEVENT](#))
A pointer to a [PARM_ENEVENT](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

- NO_ERROR**
Success.
- ERROR_INVALID_FUNCTION**
Invalid function requested.
- ERROR_INVALID_EVENT**
Invalid event handle.
- ERROR_INVALID_HID**
Invalid handle ID.
- ERROR_INVALID_EVCB**
Invalid event control block.
- ERROR_TOO_MANY_EVENTS**
Tried to enable too many events.
- FAILURE**
Stream-handler-specific error return code.

SHC_ENABLE_EVENT - Remarks

Command the stream handler to enable generation of a specific explicit event. The actual events should be reported through the [SMH_REPORT_EVENT](#) message to the Sync/Stream Manager. Only explicit events can be enabled or disabled.

Following is a list of the system-defined explicit events that can be enabled:

- [EVENT_CUE_TIME](#)
- [EVENT_CUE_TIME_PAUSE](#)
- [EVENT_CUE_DATA](#)
- [EVENT_SYNCOVERRUN](#)
- [EVENT_DATAOVERRUN](#)
- [EVENT_DATAUNDERRUN](#)

SHC_ENABLE_EVENT - Related Messages

- [SpiEnableEvent](#)

SHC_ENABLE_EVENT - Topics

- Select an item:
- [Description](#)
 - [Returns](#)
 - [Remarks](#)
 - [Related Messages](#)
 - [Glossary](#)

SHC_ENABLE_SYNC

SHC_ENABLE_SYNC Parameter - pParmln

pParmln ([PPARM_ENSYNC](#))
A pointer to a [PARM_ENSYNC](#) data structure.

SHC_ENABLE_SYNC Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_MMTIME	Invalid MMTIME specified or is the wrong type for the stream master.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_FLAG	The flag passed is invalid.
ERROR_STREAM_NOTMASTER	The stream handle passed was not a synchronization master stream.
ERROR_INVALID_NUMSLAVES	Invalid number of slave streams.
FAILURE	Stream handler-specific error return code.

SHC_ENABLE_SYNC - Description

This message enables synchronization for a stream handler in a sync group.

pParmln (**PPARM_ENSYNC**)

A pointer to a **PARM_ENSYNC** data structure.

rc (**ULONG**)

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_MMTIME

Invalid **MMTIME** specified or is the wrong type for the stream master.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_FLAG

The flag passed is invalid.

ERROR_STREAM_NOTMASTER

The stream handle passed was not a synchronization master stream.

ERROR_INVALID_NUMSLAVES

Invalid number of slave streams.

FAILURE

Stream handler-specific error return code.

SHC_ENABLE_SYNC - Remarks

Commands a stream handler to be either a master stream handler (generates sync pulses) or a slave stream handler (receives sync pulses) in a synchronized-stream group. Only one stream handler from each stream can be a master or slave in a synchronized-stream group. Only one stream handler will ever be the master. All slave stream handlers must return a **SYNC_EVCB** to be used for the sync-pulse notification. Optionally, the stream handler also can provide a system semaphore that is posted by the Sync/Stream Manager during a sync-pulse notification. Otherwise, the stream handler must poll the **EVCB_SYNC POLLING** bit in the **SYNC_EVCB** to check for sync notification from the Sync/Stream Manager. When a sync notification occurs, the *ulStatus* field of the **SYNC_EVCB** contains the sync time.

Sync pulses are distributed by the Sync/Stream Manager, based on the stream group. This distribution is effective for both DLL and device driver slave stream handlers. Device driver stream handlers receive sync pulses through their **SYNC_EVCB**. Each slave stream handler must the sync pulse **SYNC_EVCB** regularly with its calculated stream time. The Sync/Stream Manager checks the slave-handler stream time against the master stream time and determines whether to send a sync pulse to this handler.

Device driver stream handlers check for sync pulses from the Sync/Stream Manager by polling a flag in the sync-pulse **SYNC_EVCB**. The Sync/Stream Manager sets the flag to indicate a sync pulse and updates the current master-handler stream time. Typically, the device driver slave handler polls the flag once during interrupt processing and adjusts the stream consumption accordingly.

DLL stream handlers receive sync pulses in one of two ways: by registering a semaphore with the Sync/Stream Manager, or by the same method the Sync/Stream Manager uses for device driver stream handlers. DLL stream handlers then can process the sync pulse in the context of a time-critical thread (created by the stream handler at stream creation time).

The optional semaphore handle returned from the stream handler must be a 16-bit system semaphore instead of 32-bit event or 32-bit mutex semaphore because of a limitation of the device driver model. Currently, device drivers can use only 16-bit semaphores.

SHC_ENABLE_SYNC - Related Messages

- [SpiEnableSync](#)

SHC_ENABLE_SYNC - Topics

Select an item:

[Description](#)

[Returns](#)

[Remarks](#)

[Related Messages](#)

[Glossary](#)

SHC_ENUMERATE_PROTOCOLS

SHC_ENUMERATE_PROTOCOLS Parameter - pParmln

pParmln ([PPARM_ENUMPROT](#))

A pointer to a [PARM_ENUMPROT](#) data structure.

SHC_ENUMERATE_PROTOCOLS Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_SPCBKEY

Invalid [SPCBKEY](#). spcbkey.

ERROR_INVALID_BUFFER_SIZE

The buffer passed in the function is not big enough for the results. *pulNumSPCBKeys* contains the number of SPCB keys that are returned.

FAILURE

Stream handler-specific error return code.

SHC_ENUMERATE_PROTOCOLS - Description

This message returns a list of stream protocol keys for a specified stream handler.

pParmIn ([PPARM_ENUMPROT](#))
A pointer to a [PARM_ENUMPROT](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_SPCBKEY	Invalid SPCBKEY . spcbkey.
ERROR_INVALID_BUFFER_SIZE	The buffer passed in the function is not big enough for the results. <i>puNumSPCBKeys</i> contains the number of SPCB keys that are returned.
FAILURE	Stream handler-specific error return code.

SHC_ENUMERATE_PROTOCOLS - Related Messages

- [SpiEnumerateProtocols](#)

SHC_ENUMERATE_PROTOCOLS - Topics

Select an item:
[Description](#)
[Returns](#)
[Related Messages](#)
[Glossary](#)

SHC_GET_PROTOCOL

SHC_GET_PROTOCOL Parameter - pParmIn

pParmIn ([PPARM_GPROT](#))
A pointer to a [PARM_GPROT](#) data structure.

SHC_GET_PROTOCOL Return Value - rc

rc (ULONG)

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_SPCBKEY

Invalid *spcbkey*.

FAILURE

Stream-handler-specific error return code.

SHC_GET_PROTOCOL - Description

This message queries a stream handler for a specific stream protocol.

pParmIn (PPARM_GPROT)

A pointer to a [PARM_GPROT](#) data structure.

rc (ULONG)

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_SPCBKEY

Invalid *spcbkey*.

FAILURE

Stream-handler-specific error return code.

SHC_GET_PROTOCOL - Remarks

Commands the stream handler to retrieve the [SPCB](#) specified by the *spcbkey*. The stream handler copies the [SPCB](#) structure into the

supplied buffer.

SHC_GET_PROTOCOL - Related Messages

- [SpiGetProtocol](#)

SHC_GET_PROTOCOL - Topics

Select an item:
[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

SHC_GET_TIME

SHC_GET_TIME Parameter - pParmln

pParmln ([PPARM_GTIME](#))
A pointer to a [PARM_GTIME](#) data structure.

SHC_GET_TIME Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_HID	Invalid handler ID.

FAILURE
Stream-handler-specific error return code.

SHC_GET_TIME - Description

This message queries the current stream time.

pParmIn ([PPARM_GTIME](#))
A pointer to a [PARM_GTIME](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR
Success.

ERROR_INVALID_FUNCTION
Invalid function requested.

ERROR_INVALID_STREAM
Invalid stream handle.

ERROR_INVALID_HID
Invalid handler ID.

FAILURE
Stream-handler-specific error return code.

SHC_GET_TIME - Remarks

Commands the stream handler to return the current stream time for this stream. The [SPCB](#) for this stream instance indicates whether the source stream handler or target stream handler can return the current stream time. When the stream handler returns the current stream time for this stream (if this is a slave stream handler), it is not necessarily the same as the master stream time.

SHC_GET_TIME - Related Messages

- [SpiGetTime](#)
-

SHC_GET_TIME - Topics

Select an item:
[Description](#)
[Returns](#)

SHC_INSTALL_PROTOCOL

SHC_INSTALL_PROTOCOL Parameter - pParmIn

pParmIn ([PPARM_INSTPROT](#))
A pointer to a [PARM_INSTPROT](#) data structure.

SHC_INSTALL_PROTOCOL Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_SPCBKEY	Invalid <i>spcbkey</i> .
ERROR_INVALID_PROTOCOL	Invalid stream protocol control block.
ERROR_ALLOC_RESOURCES	Out of memory or some other resource.
FAILURE	Stream handler-specific error return code.

SHC_INSTALL_PROTOCOL - Description

This message installs or removes a specified stream protocol for a stream handler.

pParmIn ([PPARM_INSTPROT](#))
A pointer to a [PARM_INSTPROT](#) data structure.

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_SPCBKEY	Invalid <i>spcbkey</i> .
ERROR_INVALID_PROTOCOL	Invalid stream protocol control block.
ERROR_ALLOC_RESOURCES	Out of memory or some other resource.
FAILURE	Stream handler-specific error return code.

SHC_INSTALL_PROTOCOL - Remarks

The stream handler is required to support this message because [SpiGetHandler](#) uses this message to install the default [SPCB](#) in the stream handler. The stream handler must support the installation and removal of [SPCBs](#), but not the replacement of an [SPCB](#). The stream handler fails an install if the [SPCB](#) exists. The Sync/Stream Manager fails any installation or removal of [SPCBs](#) with *ullntKey* equal to 0. This prevents one process from affecting the stream creation of another process.

SHC_INSTALL_PROTOCOL - Related Messages

- [SpiInstallProtocol](#)

SHC_INSTALL_PROTOCOL - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

SHC_NEGOTIATE_RESULT

SHC_NEGOTIATE_RESULT Parameter - pParmIn

pParmIn ([PPARM_NEGOTIATE](#))

A pointer to a [PARM_NEGOTIATE](#) data structure.

SHC_NEGOTIATE_RESULT Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_PROTOCOL

There was an [SPCB](#) negotiation failure. *ulErrorStatus* points to the field of the [SPCB](#) that failed the negotiation. The fields of the [SPCB](#) are numbered, starting with 1.

FAILURE

Stream handler-specific error return code.

SHC_NEGOTIATE_RESULT - Description

This message provides the results of an [SPCB](#) negotiation for a stream instance.

pParmIn ([PPARM_NEGOTIATE](#))

A pointer to a [PARM_NEGOTIATE](#) data structure.

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_PROTOCOL

There was an [SPCB](#) negotiation failure. *ulErrorStatus* points to the field of the [SPCB](#) that failed the negotiation. The fields of the [SPCB](#) are numbered, starting with 1.

FAILURE

Stream handler-specific error return code.

SHC_NEGOTIATE_RESULT - Remarks

Notifies the stream handler that the [SPCB](#) negotiation is complete, and gives the negotiated [SPCB](#) contents to the handler.

SHC_NEGOTIATE_RESULT - Related Messages

- [SpiCreateStream](#)

SHC_NEGOTIATE_RESULT - Topics

Select an item:
[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

SHC_SEEK

SHC_SEEK Parameter - pParmln

pParmln ([PPARM_SEEK](#))
A pointer to a [PARM_SEEK](#) data structure.

SHC_SEEK Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

NO_ERROR	Success.
--------------------------	----------

ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_MMTIME	The specific MMTIME format is not understood by this stream.
ERROR_INVALID_HID	Invalid handler ID.
ERROR_INVALID_FLAG	The flag passed is invalid.
ERROR_DATA_ITEM_NOT_SPECIFIED	The data object was not specified.
ERROR_STREAM_NOT_SEEKABLE	Seek is an invalid request for this stream. (For example, performing a seek while recording.)
ERROR_DATA_ITEM_NOT_SEEKABLE	Seek is an invalid request for the data object.
ERROR_NOT_SEEKABLE_BY_TIME	Cannot seek the data object using an MMTIME value.
ERROR_NOT_SEEKABLE_BY_BYTES	Cannot seek the data object using a byte value.
ERROR_STREAM_NOT_STOP	The stream cannot perform the request.
FAILURE	Stream handler-specific error return code.

SHC_SEEK - Description

This message seeks to a specified point in the stream object or sets the current stream time.

pParmln ([PPARM_SEEK](#))

A pointer to a [PARM_SEEK](#) data structure.

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR	Success.
ERROR_INVALID_FUNCTION	Invalid function requested.
ERROR_INVALID_STREAM	Invalid stream handle.
ERROR_INVALID_MMTIME	The specific MMTIME format is not understood by this stream.
ERROR_INVALID_HID	Invalid handler ID.

ERROR_INVALID_FLAG	The flag passed is invalid.
ERROR_DATA_ITEM_NOT_SPECIFIED	The data object was not specified.
ERROR_STREAM_NOT_SEEKABLE	Seek is an invalid request for this stream. (For example, performing a seek while recording.)
ERROR_DATA_ITEM_NOT_SEEKABLE	Seek is an invalid request for the data object.
ERROR_NOT_SEEKABLE_BY_TIME	Cannot seek the data object using an MMTIME value.
ERROR_NOT_SEEKABLE_BY_BYTES	Cannot seek the data object using a byte value.
ERROR_STREAM_NOT_STOP	The stream cannot perform the request.
FAILURE	Stream handler-specific error return code.

SHC_SEEK - Remarks

Commands the stream handler to seek to a specified point in the stream. The */SeekPoint* can be represented in [MMTIME](#) format.

The */SeekPoint* can be passed as an absolute or a relative seek position. The stream handler must return the absolute seek point in this field. Stream handlers that perform physical device seeks receive the SPI_SEEK_FROMEND flag if it was passed on the [SpiSeekStream](#) call from the application or media control device. Stream handlers that do not perform physical device seeks do not receive this flag. A stream handler must specify in the stream protocol whether it performs a physical seek.

SHC_SEEK - Related Messages

- [SpiSeekStream](#)

SHC_SEEK - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

SHC_SENDMSG

SHC_SENDMSG Parameter - pParmln

pParmln ([PPARM_SNDMSG](#))

A pointer to a [PARM_SNDMSG](#) data structure.

SHC_SENDMSG Return Value - rc

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_HID

Invalid handler ID.

FAILURE

Stream handler-specific error return code.

SHC_SENDMSG - Description

This message sends a stream handler specific message to a stream handler.

pParmln ([PPARM_SNDMSG](#))

A pointer to a [PARM_SNDMSG](#) data structure.

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_HID

Invalid handler ID.

FAILURE

Stream handler-specific error return code.

SHC_SENDMSG - Remarks

This message allows an application to communicate with a stream handler directly. It is used to pass information during active streaming of data. It is similar to the [SHC_ASSOCIATE](#) message, but it does not associate an object with a stream and is usable while the stream is running. The message control block is stream handler specific and the interface must be provided by the stream handler.

SHC_SENDMSG - Topics

- Select an item:
- [Description](#)
 - [Returns](#)
 - [Remarks](#)
 - [Glossary](#)

SHC_START

SHC_START Parameter - pParmln

pParmln ([PPARM_START](#))
A pointer to a [PARM_START](#) data structure.

SHC_START Return Value - rc

- rc** ([ULONG](#))
Return codes indicating success or the type of failure:
- | | |
|--------------------------------------|--|
| NO_ERROR | Success. |
| ERROR_INVALID_FUNCTION | Invalid function requested. |
| ERROR_INVALID_STREAM | Invalid stream handle. |
| ERROR_INVALID_HID | Invalid handler ID. |
| ERROR_DATA_ITEM_NOT_SPECIFIED | Data object was not specified. |
| ERROR_DEVICE_NOT_FOUND | The device was not found. |
| FAILURE | Stream handler-specific error return code. |

SHC_START - Description

This message starts data streaming for a stream handler of a particular stream instance.

pParmln ([PPARM_START](#))

A pointer to a [PARM_START](#) data structure.

rc ([ULONG](#))

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_DATA_ITEM_NOT_SPECIFIED

Data object was not specified.

ERROR_DEVICE_NOT_FOUND

The device was not found.

FAILURE

Stream handler-specific error return code.

SHC_START - Remarks

Commands the stream handler to either start producing data for the stream, if it is a source, or start consuming data from the stream if it is a target.

SHC_START - Related Messages

- [SpiStartStream](#)
-

SHC_START - Topics

Select an item:

[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

SHC_STOP

SHC_STOP Parameter - pParmln

pParmln ([PPARM_STOP](#))
A pointer to a [PARM_STOP](#) data structure.

SHC_STOP Return Value - rc

rc ([ULONG](#))
Return codes indicating success or the type of failure:

<code>NO_ERROR</code>	Success.
<code>ERROR_INVALID_FUNCTION</code>	Invalid function requested.
<code>ERROR_INVALID_STREAM</code>	Invalid stream handle.
<code>ERROR_INVALID_HID</code>	Invalid handler ID.
<code>ERROR_INVALID_FLAG</code>	The flag passed is invalid.
<code>ERROR_STREAM_NOT_STARTED</code>	The stream cannot perform the requested message unless the stream has been started.
<code>FAILURE</code>	Stream-handler-specific error return code.

SHC_STOP - Description

This message stops data streaming for a stream handle particular stream instance.

pParmIn (PPARM_STOP)

A pointer to a [PARM_STOP](#) data structure.

rc (ULONG)

Return codes indicating success or the type of failure:

NO_ERROR

Success.

ERROR_INVALID_FUNCTION

Invalid function requested.

ERROR_INVALID_STREAM

Invalid stream handle.

ERROR_INVALID_HID

Invalid handler ID.

ERROR_INVALID_FLAG

The flag passed is invalid.

ERROR_STREAM_NOT_STARTED

The stream cannot perform the requested message unless the stream has been started.

FAILURE

Stream-handler-specific error return code.

SHC_STOP - Remarks

Commands the stream handler to stop streaming data. The flags will indicate the type of stop (discard, flush, pause) initiated through SPI. The *pause stop* immediately pauses the stream by sending an SHC_STOP message to both the source and target stream handlers. Both stream handlers must shut down and not request any more buffers from the Sync/Stream Manager. The Sync/Stream Manager fails any *get buffer* requests, so that a stream handler cannot continue to stream. All stream data remains valid. The stream can be restarted by issuing [SpiStartStream](#).

The *discard stop* immediately stops the stream by sending an SHC_STOP message to both the source and target stream handlers. Both stream handlers must shut down and not request any more buffers from the Sync/Stream Manager. But they must return all buffers to the Sync/Stream Manager. The Sync/Stream Manager fails any *get buffer* requests, so that a stream handler cannot continue to stream. All stream data is discarded. The stream can be restarted, but the data in the discarded buffers is lost.

The *flush stop* sends an SHC_STOP message to both the source and target stream handlers. The stream does not stop, however. The source stream handler must shut down and not request any more buffers from the Sync/Stream Manager. It must return any buffers to the Sync/Stream Manager. The Sync/Stream Manager fails any *get buffer* requests from the source stream handler, so that it cannot continue to fill buffers. The target stream handler continues to stream data until all buffers are empty. The stream can be restarted with no loss of data.

For discard and flush stops, the Sync/Stream Manager detects when the stream handlers have stopped and returned all buffers. At this point, the Sync/Stream Manager notifies the application media control device with an EVENT_STREAM_STOPPED message. Also, both stream handlers must return all owned buffers to the Sync/Stream Manager before the stream EVENT_STREAM_STOPPED message is sent to the application media control device.

If a stream is paused and the application issues a discard stop, the stream buffers is discarded and the stream is put in a stopped state. If a stream is paused and a flush stop is issued, the remaining stream buffers is transferred to the target stream handler. In other words, the stream begins *playing* again.

Slave streams can be stopped independently of the master stream. When a slave stream stops, the synchronized relationship is not broken, but the slave stream is quiesced. The master stream and any other slaves continue to stream. If the stopped slave stream is restarted, a new master-to-slave time offset is established which the slave stream maintains until it is either stopped again, or the master is stopped. This enables slave streams to regain synchronization at any asynchronous point in time, for example, in response to another stream event from the master. Slave stream time and master-stream time are not always identical. Once a slave stream is stopped, the master stream time can continue to increment. When the slave stream is restarted, its stream time increments at the same rate as the master stream. Any slave in a synchronized relationship can be arbitrarily started or stopped without affecting the activity of the master stream or any other slave stream. When the SPI_STOP_SLAVES flag is set, the master streams, as well as the slave streams stop.

Note: Typically, data streaming continues until the end of the stream. Then the Sync/Stream Manager sends an EVENT_EOS message to the application media control device's event routine. The Sync/Stream Manager detects EVENT_EOS and EVENT_STREAM_STOPPED and notifies the application. Therefore, the stream handler does not report these events to the Sync/Stream Manager.

SHC_STOP - Related Messages

- [SpiStopStream](#)

SHC_STOP - Topics

Select an item:
[Description](#)
[Returns](#)
[Remarks](#)
[Related Messages](#)
[Glossary](#)

Data Types

This section describes data types in C language. A data type name beginning with "P" (for example, PMCI_CONNECTOR_PARMS) is likely to be a pointer to another data type (in this instance, MCI_CONNECTOR_PARMS). If no data type definition can be found in this section for a data type name "Pxxxxxx," it becomes a pointer to the data type "xxxxxx," for which a definition should be found in this section. The implicit type definition needed for such a pointer "Pxxxxxx" is:

```
typedef xxxxxx *Pxxxxxx;
```

Such definitions are provided by means of the system header files.

ACB

Associate control block. This structure describes a data object to associate with a stream instance. Each stream handler defines the values for *ulParm1* and *ulParm2*. A stream handler can define none or more associate control blocks.

```
typedef struct _ACB {
    ULONG      ulACBLen;    /* Structure length. */
    ULONG      ulObjType;   /* Object type. */
    ULONG      ulParm1;     /* I/O parameter. */
    ULONG      ulParm2;     /* I/O parameter. */
} ACP;

typedef ACP *PACP;
```

ACB Field - ulACBLen

ulACBLen (ULONG)
Length of the associate control block.

ACB Field - ulObjType

ulObjType (ULONG)
Identifies the type of object being associated with a stream handler. Each stream handler defines a set of object types that it accepts.

ACB Field - ulParm1

ulParm1 (ULONG)
Input or output parameter. This is defined by the stream handler for each object type.

ACB Field - ulParm2

ulParm2 (ULONG)
Input or output parameter. This is defined by the stream handler for each object type.

ACB_CDDA

CD Associate Control Block.

```
typedef struct _ACB_CDDA {
    ULONG      ulACBLen;    /* Structure length. */
    ULONG      ulObjType;   /* ACBTYPE_CDDA. */
    CHAR       bcDDrive;    /* CD drive letter. */
} ACD_CDDA;

typedef ACD_CDDA *PACD_CDDA;
```

ACB_CDDA Field - ulACBLen

ulACBLen ([ULONG](#))
Length of the associate control block.

ACB_CDDA Field - ulObjType

ulObjType ([ULONG](#))
Identifies the type of object being associated with a stream handler. Each stream handler defines a set of object types that it accepts.

ACB_CDDA Field - bCDDrive

bCDDrive ([CHAR](#))
Specifies the letter of the CD drive.

ACB_CODECSH

This is the Associate Control Block data structure for the CODEC stream handler object.

```
typedef struct _ACB_CODECSH {  
    ULONG      ulACBLen;          /* Structure length. */  
    ULONG      ulObjType;         /* ACBTYPE_CODECSH. */  
    HSTREAM     hstreamToPair;    /* Second stream of pair. */  
    PVOID       pMmioInfo;        /* Information for IOProc. */  
    ULONG      ulInfoLength;      /* Length of MmioInfo. */  
    PVOID       pCodecControl;    /* CODEC control information. */  
    ULONG      ulControlLength;   /* CodecControl length. */  
} ACB_CODECSH;  
  
typedef ACB_CODECSH *PACB_CODECSH;
```

ACB_CODECSH Field - ulACBLen

ulACBLen ([ULONG](#))
Length of the structure.

ACB_CODECSH Field - ulObjType

ulObjType ([ULONG](#))
Defines the object type as ACBTYPE_CODECSH.

ACB_CODECSH Field - hstreamToPair

hstreamToPair ([HSTREAM](#))
Second stream of pair.

ACB_CODECSH Field - pMmiolInfo

pMmiolInfo ([PVOID](#))
Information for IOProc.

ACB_CODECSH Field - ullInfoLength

ullInfoLength ([ULONG](#))
Length of MmiolInfo.

ACB_CODECSH Field - pCodecControl

pCodecControl ([PVOID](#))
CODEC control information.

ACB_CODECSH Field - ulControlLength

ulControlLength ([ULONG](#))
CodecControl length.

ACB_MEM_PLAYL

This structure is a pointer to a memory stream handler playlist object ACB.

```
typedef struct _ACB_MEM_PLAYL {
    ULONG    ulACBLen;        /* Structure length. */
    ULONG    ulObjType;      /* Object type. */
    PVOID    pMemoryAddr;    /* Memory object starting address. */
} ACB_MEM_PLAYL;

typedef ACB_MEM_PLAYL *PACB_MEM_PLAYL;
```

ACB_MEM_PLAYL Field - ulACBLen

ulACBLen (ULONG)
Length of the associate control block structure.

ACB_MEM_PLAYL Field - ulObjType

ulObjType (ULONG)
Defines the object type as ACBTYPE_MEM_PLAYL.

ACB_MEM_PLAYL Field - pMemoryAddr

pMemoryAddr (PVOID)
Starting address of the memory object.

ACB_MISH

This is the MIDI Associate Control Block data structure.

```
typedef struct _ACB_MISH {
    ULONG    ulACBLen;        /* Structure length. */
    ULONG    ulObjType;      /* Object type. */
    HSTREAM  hstreamDefault;  /* Default HSTREAM. */
    ULONG    ulDeviceTypeID;  /* Device type ID. */
    ULONG    ulpMapperPorts;  /* Pointer to mapper port table. */
    ULONG    ulNumInStreams;  /* Number of input streams. */
    HSTREAM  hstreamIn[MAX_PORTS]; /* Input stream array. */
    ULONG    ulNumOutStreams;  /* Index to output array. */
    HSTREAM  hstreamOut[MAX_PORTS]; /* Output stream array. */
} ACB_MISH;

typedef ACB_MISH *PACB_MISH;
```

ACB_MISH Field - ulACBLen

ulACBLen ([ULONG](#))
Length of the structure.

ACB_MISH Field - ulObjType

ulObjType ([ULONG](#))
Defines the object type as ACBTYPE_MISH.

ACB_MISH Field - hstreamDefault

hstreamDefault ([HSTREAM](#))
The default HSTREAM to use when the mapper is turned off.

ACB_MISH Field - ulDeviceTypeID

ulDeviceTypeID ([ULONG](#))
Describes the device type ID.

ACB_MISH Field - ulpMapperPorts

ulpMapperPorts ([ULONG](#))
Pointer to mapper port table.

ACB_MISH Field - ulNumInStreams

ulNumInStreams ([ULONG](#))

The number of input streams.

ACB_MISH Field - hstreamIn[MAX_PORTS]

hstreamIn[MAX_PORTS] ([HSTREAM](#))
The input stream array.

ACB_MISH Field - ulNumOutStreams

ulNumOutStreams ([ULONG](#))
Lists objects in the output array.

ACB_MISH Field - hstreamOut[MAX_PORTS]

hstreamOut[MAX_PORTS] ([HSTREAM](#))
Output stream array. The index into the array is the source channel for that stream.

ACB_MMIO

This structure contains fields for a MMIO associate control block.

```
typedef struct _ACB_MMIO {  
    ULONG      ulACBLen;    /* Length of structure. */  
    ULONG      ulObjType;   /* Object type. */  
    HMMIO      hmmio;       /* Handle of media element manager object. */  
} ACP_MMIO;  
  
typedef ACP_MMIO FAR *PACP_MMIO;
```

ACB_MMIO Field - ulACBLen

ulACBLen ([ULONG](#))
Length of the associate control block structure.

ACB_MMIO Field - ulObjType

ulObjType (ULONG)
Object type.

ACB_MMIO Field - hmmio

hmmio (HMMIO)
Handle of media element manager object.

ACB_MTSH

This is the Associate Control Block data structure for the multitrack stream handler object.

```
typedef struct _ACB_MTSH {
    ULONG      ulACBLen;      /* Structure length. */
    ULONG      ulObjType;     /* ACBTYPE_MTSH. */
    HMMIO      hmmio;        /* Handle of media element manager object. */
    MMTRACKINFO mmtrackInfo;  /* Track for this stream. */
    ULONG      ulFlags;       /* Flags. */
} ACP_MTSH;

typedef ACP_MTSH *PACP_MTSH;
```

ACB_MTSH Field - ulACBLen

ulACBLen (ULONG)
Length of the structure.

ACB_MTSH Field - ulObjType

ulObjType (ULONG)
Defines the object type as ACBTYPE_MTSH.

ACB_MTSH Field - hmmio

hmmio ([HMMIO](#))
Handle of media element manager object.

ACB_MTSH Field - mmtrackInfo

mmtrackInfo ([MMTRACKINFO](#))
Track for this stream.

ACB_MTSH Field - ulFlags

ulFlags ([ULONG](#))
Flags.

ACB_NULLSH

This data structure allows users of the NULL stream handler to register a callback address. This callback address is used to pass information from the NULL stream handler to the user. There are two stream handler commands (SHC messages) that the NULL stream handler passes to the user ([SHC_ENABLE_SYNC](#) and [SHC_DISABLE_SYNC](#)). These are required to allow the user to pass back to the Sync/Stream Manager an [EVCB](#) and an optional semaphore to be used for synchronization.

The prototype for the callback address is:

```
ULONG pfnEntry(PVOID pParmin)
```

where *pParmin* is a pointer to an SHC message-specific parameter block.

```
typedef struct _ACB_NULLSH {  
    ULONG    ulADBLen;    /* Structure length. */  
    ULONG    ulObjType;   /* Object type. */  
    PFN      pfnEntry;    /* User callback entry point. */  
    ULONG    ulReserved;  /* Reserved. */  
} ACB_NULLSH;
```

```
typedef ACB_NULLSH *PACB_NULLSH;
```

ACB_NULLSH Field - ulADBLen

ulADBLen ([ULONG](#))
Length of the structure.

ACB_NULLSH Field - ulObjType

ulObjType ([ULONG](#))

Defines the object type as ACBTYPE_NULLSH and identifies the type of object being associated with the stream handler. Each stream handler defines a set of object types that it accepts.

ACB_NULLSH Field - pfnEntry

pfnEntry ([PFN](#))

Specifies the user callback address.

ACB_NULLSH Field - ulReserved

ulReserved ([ULONG](#))

Reserved.

ACB_SET

This is the Set Associate Control Block data structure.

```
typedef struct _ACB_SET {  
    ULONG    ulACBLen;    /* Structure length. */  
    ULONG    ulObjType;   /* ACBTYPE_SET. */  
    ULONG    ulFlags;     /* Set flags. */  
    ULONG    ulReserved;  /* Reserved. */  
} ACB_SET;  
  
typedef ACB_SET *PACB_SET;
```

ACB_SET Field - ulACBLen

ulACBLen ([ULONG](#))

Length of the structure.

ACB_SET Field - ulObjType

ulObjType (ULONG)

Defines the object type as ACBTYPE_SET.

ACB_SET Field - ulFlags

ulFlags (ULONG)

Set flags.

ACB_SET Field - ulReserved

ulReserved (ULONG)

Reserved.

BITMAPFILEHEADER2

Bit-map file header structure.

```
typedef struct _BITMAPFILEHEADER2 {  
    USHORT    usType;      /* Type of resource the file contains. */  
    ULONG     cbSize;      /* Size of the BITMAPFILEHEADER2 structure in bytes. */  
    SHORT     xHotspot;    /* Width of hotspot for icons and pointers. */  
    SHORT     yHotspot;    /* Height of hotspot for icons and pointers. */  
    ULONG     offBits;     /* Offset in bytes. */  
    BITMAPINFOHEADER2 bmp2; /* Bit-map information header structure. */  
} BITMAPFILEHEADER2;
```

```
typedef BITMAPFILEHEADER2 *PBITMAPFILEHEADER2;
```

BITMAPFILEHEADER2 Field - usType

usType (USHORT)

The valid values are:

BFT_BMAP

(0x4D42 - 'BM' for bit maps)

BFT_ICON

(0x4349 - 'IC' for icons)

BFT_POINTER
(0x4540 - 'PT' for pointers)

BFT_COLORICON
(0x4943 - 'CI' for color icons)

BFT_COLORPOINTER
(0x5043 - 'CP' for color pointers)

BITMAPFILEHEADER2 Field - cbSize

cbSize ([ULONG](#))
Size of the BITMAPFILEHEADER2 structure in bytes.

BITMAPFILEHEADER2 Field - xHotspot

xHotspot ([SHORT](#))
The X-coordinate of the hotspot for icons and pointer. This field is ignored for bit maps.

BITMAPFILEHEADER2 Field - yHotspot

yHotspot ([SHORT](#))
The Y-coordinate of the hotspot for icons and pointer. This field is ignored for bit maps.

BITMAPFILEHEADER2 Field - offBits

offBits ([ULONG](#))
The offset in bytes to beginning of bit-map pel data in the file, from the start of the definition.

BITMAPFILEHEADER2 Field - bmp2

bmp2 ([BITMAPINFOHEADER2](#))
Bit-map information header structure.

BITMAPINFOHEADER2

This structure contains size, color type, extent, and the palette information for an image.

```
typedef struct _BITMAPINFOHEADER2 {
    ULONG      cbFix;           /* Size of header. */
    ULONG      cx;              /* Bitmap width. */
    ULONG      cy;              /* Bitmap height. */
    USHORT     cPlanes;         /* Number of planes. */
    USHORT     cBitCount;       /* Bits per pel. */
    ULONG      ulCompression;    /* Compression scheme for storing bitmap. */
    ULONG      cbImage;         /* Bitmap storage data length. */
    ULONG      cxResolution;     /* Horizontal resolution of target device. */
    ULONG      cyResolution;     /* Vertical resolution of target device. */
    ULONG      cClrUsed;         /* Number of color indexes used. */
    ULONG      cClrImportant;    /* Number of color indexes. */
    USHORT     usUnits;         /* Units of measure for resolution. */
    USHORT     usReserved;       /* Reserved. Must be 0. */
    USHORT     usRecording;      /* Must be BRA_BOTTOMUP. */
    USHORT     usRendering;      /* Not used. */
    ULONG      cSize1;           /* Size value field 1. */
    ULONG      cSize2;           /* Size value field 2. */
    ULONG      ulColorEncoding;  /* Color encoding. */
    ULONG      ulIdentifier;     /* Reserved. */
} BITMAPINFOHEADER2;

typedef BITMAPINFOHEADER2 *PBITMAPINFOHEADER2;
```

BITMAPINFOHEADER2 Field - cbFix

cbFix ([ULONG](#))
Specifies the size of this header from this field forward.

BITMAPINFOHEADER2 Field - cx

cx ([ULONG](#))
Specifies the width of the bitmap, in pels.

BITMAPINFOHEADER2 Field - cy

cy ([ULONG](#))
Specifies the height of the bitmap, in pels.

BITMAPINFOHEADER2 Field - cPlanes

cPlanes ([USHORT](#))
Specifies the number of planes.

BITMAPINFOHEADER2 Field - cBitCount

cBitCount ([USHORT](#))
Specifies the number of bits per pel. Valid values are 1, 4, 8, or 24.

BITMAPINFOHEADER2 Field - ulCompression

ulCompression ([ULONG](#))
Compression scheme used to store the bitmap. The valid value for the standard presentation format is `BCA_UNCOMP = 0xFFFF0000`, for uncompressed data.

BITMAPINFOHEADER2 Field - cbImage

cbImage ([ULONG](#))
Length of bitmap storage data, in bytes. This field should match the value in *ulMemSize* in the [XDIBHDR_PREFIX](#) structure.

BITMAPINFOHEADER2 Field - cxResolution

cxResolution ([ULONG](#))
Horizontal resolution X component of the target device that the bitmap is intended for, in the units specified in the *usUnits* field. This information, along with the Y resolution, enables applications to pick out a bitmap from a resource group to best fit the characteristics of the current output device.

BITMAPINFOHEADER2 Field - cyResolution

cyResolution ([ULONG](#))
Vertical resolution Y component of the target device that the bitmap is intended for, in the units specified by the *usUnits* field. This

information, along with the X resolution, enables applications to pick out a bitmap from a resource group to best fit the characteristics of the current output device.

BITMAPINFOHEADER2 Field - cclrUsed

cclrUsed (ULONG)

Number of color indexes used. A 0 value (default) means all the colors are used. For a nonzero value, only the first *cclrUsed* entries in the color table are accessed. The rest can be omitted.

For standard formats, with a *cBitCount* of 1, 4, or 8, any indexes beyond the value in this field are invalid. For example, a bitmap with 64 colors can use the 8-bit count format without having to supply the other 192 entries in the color table. For the standard 24-bit count, this field contains the number of colors used by the bitmap.

BITMAPINFOHEADER2 Field - cclrImportant

cclrImportant (ULONG)

Number of important color indexes. The first *cclrImportant* color entries are important. A zero (default) means that all are important. For a 24-bit count standard, these colors also are listed in the *uiColorEncoding* color array.

This information helps to make palette management more responsive to an application's needs. The important colors are more likely to be assigned to the device palette, while others might be mapped to the nearest colors available.

BITMAPINFOHEADER2 Field - usUnits

usUnits (USHORT)

Units of measure for the horizontal and vertical resolution. BRU_METRIC, or pels per meter, is the default value.

BITMAPINFOHEADER2 Field - usReserved

usReserved (USHORT)

Reserved for future use and must be set to zero.

BITMAPINFOHEADER2 Field - usRecording

usRecording (USHORT)

Recording algorithm used. This field must be set to BRA_BOTTOMUP. Scan lines will be handled in bottom-to-top order.

BITMAPINFOHEADER2 Field - usRendering

usRendering (USHORT)
This field is currently not used. Its purpose is for the specification of a halftoning algorithm. The algorithm is used to record bitmap data that has been digitally halftoned. Valid values might include:

BRH_NOTHALFTONED
 Bitmap data is not halftoned. This is the default.

BRH_ERRORDIFFUSION
 Error-diffusion or damped error-diffusion algorithm.

BRH_PANDA
 Processing algorithm for noncoded document acquisition.

BRH_SUPERCIRCLE
 Super circle algorithm.

BITMAPINFOHEADER2 Field - cSize1

cSize1 (ULONG)
Size value field 1. If BRH_ERRORDIFFUSION is specified in the *usRendering* field, this represents the error damping as a percentage in the range 0-100%. A value of 100% indicates no damping; a value of 0% indicates that errors are not diffused.

BITMAPINFOHEADER2 Field - cSize2

cSize2 (ULONG)
Size value field 2. If BRH_ERRORDIFFUSION is specified in the *usRendering* field, this parameter is ignored. If BRH_PANDA or BRH_SUPERCIRCLE is specified, this field is the Y-dimension of the pattern used, in pels.

BITMAPINFOHEADER2 Field - ulColorEncoding

ulColorEncoding (ULONG)
Color encoding. Each element in the array is an RGB2 data type, by default.

BITMAPINFOHEADER2 Field - ullIdentifier

ulIdentifier ([ULONG](#))

Reserved for application use.

BOOL

Boolean.

Valid values are:

- FALSE, which is 0
- TRUE, which is 1

```
typedef unsigned long BOOL;
```

BYTE

A byte.

```
typedef unsigned char BYTE;
```

CHAR

Single-byte character.

```
#define CHAR char
```

CODECASSOC

This data structure is used to associate a compressor or decompressor with an MMIO handle of an open file. This structure is used with the [mmioSet](#) function or the [MMIOM_SET](#) message.

```
typedef struct _CODECASSOC {  
    PVOID                pCodecOpen;        /* Pointer to CODECOPEN. */  
    PCODECINIFILEINFO    pCODECIniFileInfo; /* Pointer to structure. */  
} CODECASSOC;  
  
typedef CODECASSOC *PCODECASSOC;
```

CODECASSOC Field - pCodecOpen

pCodecOpen (PVOID)

Pointer to a [CODECOPEN](#) structure used by the file format I/O procedure to open the CODEC procedure. If the file already exists, such as playing a file, only the *pDstHdr* field of the [CODECOPEN](#) structure is set by the application. Other information in the [CODECOPEN](#) structure must be set by the file format I/O procedure.

CODECASSOC Field - pCODECIniFileInfo

pCODECIniFileInfo (PCODECINIFILEINFO)

Pointer to the [CODECINIFILEINFO](#) structure. This structure contains information that identifies a CODEC.

CODECINIFILEINFO

This structure contains information about a CODEC entry in the MMPMMMIO.INI file. Each entry describes a CODEC that can be used system. Some CODECs might have several entries in the MMPMMMIO.INI file. This structure is used on the [mmioIniFileCODEC](#) function.

```
typedef struct _CODECINIFILEINFO {
    ULONG      ulStructLen;           /* Length of this structure. */
    FOURCC     fcc;                  /* File format ID. */
    CHAR       szDLLName[DLLNAME_SIZE]; /* DLL name string. */
    CHAR       szProcName[PROCNAME_SIZE]; /* Procedure name string. */
    ULONG      ulCompressType;        /* Compression type. */
    ULONG      ulCompressSubType;     /* Compression subtype. */
    ULONG      ulMediaType;          /* Media type. */
    ULONG      ulCapsFlags;          /* Capabilities flags. */
    ULONG      ulFlags;              /* Reserved. */
    CHAR       szHWID[CODEC_HW_NAME_SIZE]; /* Hardware adapter name. */
    ULONG      ulMaxSrcBufLen;        /* Maximum source buffer length. */
    ULONG      ulSyncMethod;          /* Synchronization method. */
    ULONG      ulReserved1;           /* Reserved. */
    ULONG      ulXalignment;          /* Byte alignment of starting address. */
    ULONG      ulYalignment;          /* Top left Y pixel alignment. */
    ULONG      ulSpecInfo[CODEC_INFO_SIZE]; /* Contains CODEC-specific info. */
} CODECINIFILEINFO;

typedef CODECINIFILEINFO *PCODECINIFILEINFO;
```

CODECINIFILEINFO Field - ulStructLen

ulStructLen (ULONG)

This field indicates the length of the CODECINIFILEINFO structure.

CODECINIFILEINFO Field - fcc

fcc ([FOURCC](#))

The character code that identifies the file format (for example, AVI).

CODECINIFILEINFO Field - szDLLName[DLLNAME_SIZE]

szDLLName[DLLNAME_SIZE] ([CHAR](#))

The DLL file name of the CODEC Proc.

CODECINIFILEINFO Field - szProcName[PROCNAME_SIZE]

szProcName[PROCNAME_SIZE] ([CHAR](#))

The entry procedure name of the CODEC Proc.

CODECINIFILEINFO Field - ulCompressType

ulCompressType ([ULONG](#))

Specifies the compression type. For example, the TIFF file format has defined 2 for CCITT G3 modified Huffman run-length encoding, 5 for LZW compression.

CODECINIFILEINFO Field - ulCompressSubType

ulCompressSubType ([ULONG](#))

Specifies the compression subtype. For example, the TIFF file format has defined 6 as the compression type for JPEG, 1 as the compression subtype for Baseline Sequential Process, and 14 as the compression subtype for Lossless Process with Huffman Coding.

CODECINIFILEINFO Field - ulMediaType

ulMediaType ([ULONG](#))

Indicates the media type of the CODEC Proc (for example, image, audio, video or plain data). Following are the currently defined types:

MMIO_MEDIATYPE_DIGITALVIDEO

Video media type.

CODECINIFILEINFO Field - ulCapsFlags

ulCapsFlags (ULONG)

Indicates the capabilities supported by the CODEC Proc. Following are capabilities are defined for video.

CODEC_COMPRESS

Can compress.

CODEC_DECOMPRESS

Can decompress.

CODEC_WINDOW_CLIPPING

Supports clipping.

CODEC_PALETTE_TRANS

Supports palette translation.

CODEC_SELFHEAL

Supports self healing data stream.

CODEC_SCALE_PEL_HALVED

Supports pel halving of the source image.

CODEC_SCALE_CONTINUOUS

Supports continuous scaling or stretching.

CODEC_MULAPERTURE

Indicates the CODEC Proc supports multi-aperture algorithm. If this bit is off, fixed aperture is assumed.

CODEC_4_BIT_COLOR

Indicates the CODEC Proc supports 16 colors.

CODEC_8_BIT_COLOR

Indicates the CODEC Proc supports 256 colors.

CODEC_16_BIT_COLOR

Indicates the CODEC Proc supports 65536 colors.

CODEC_24_BIT_COLOR

Indicates the CODEC Proc supports 16777216 colors.

CODEC_HARDWARE

Indicates the CODEC Proc is hardware assisted and the name of the hardware is contained in the *szHWID* field.

CODEC_SYMMETRIC

Indicates the CODEC Proc supports symmetric record.

CODEC_ASYMMETRIC

Indicates the CODEC Proc supports asymmetric record.

CODEC_DIRECT_DISPLAY

Indicates the CODEC Proc can directly compress into the video RAM.

CODEC_DEFAULT

Indicates this CODEC entry will be loaded as a default when multiple CODEC entries for the same algorithm are installed in the initialization file. This allows the application to select the best CODEC for performance.

CODEC_ORIGIN_LOWERLEFT

Indicates the CODEC Proc will decompress data to the destination buffer using the lower left as window origin. The compressed data lines will be stored in reverse order in the destination buffer. Must be set when CODEC_WINDOW_CLIPPING or CODEC_DIRECT_DISPLAY are not set to allow Presentation Manager to display data correctly.

CODEC_ORIGIN_UPPERLEFT

Indicates the CODEC Proc will decompress data to the destination buffer using the upper left as window origin. This means that decompressed data lines will be stored in the same order as the incoming compressed data lines.

CODEC_SET_QUALITY

Indicates the CODEC Proc supports the quality level setting. See the [MMVIDEOOPEN](#) structure for details.

CODEC_DATA_CONSTRAINT

Indicates the CODEC Proc supports the data constraint and interval setting. See the [MMVIDEOOPEN](#) structure for details.

CODECINIFILEINFO Field - ulFlags

ulFlags ([ULONG](#))

Reserved for future use and must be set to zero.

CODECINIFILEINFO Field - szHWID[CODEC_HW_NAME_SIZE]

szHWID[CODEC_HW_NAME_SIZE] ([CHAR](#))

A null-terminated hardware adapter name string associated with CODEC.

CODECINIFILEINFO Field - ulMaxSrcBufLen

ulMaxSrcBufLen ([ULONG](#))

Indicates the maximum source buffer length in bytes the CODEC Proc can handle. Zero indicates unlimited size.

CODECINIFILEINFO Field - ulSyncMethod

ulSyncMethod ([ULONG](#))

Indicates the synchronization methods supported by the CODEC Proc. The following methods are currently defined for video.

CODEC_SYNC_METHOD_NO_DROP_FRAMES

The CODEC will not be requested to drop frames. That is, the MMIO_DROP_DELTA_FRAME flag will not be set, even if video is behind. CODECs with this synchronization method will not maintain audio synchronization if there are insufficient processing resources to decompress and display data in real time. Factors contributing to this situation include computational complexity of the CODEC during decompression, hardware capabilities such as processor speed, bus speed, video I/O (memory wait state) speed, and other processes running in the system contending for processor resources at potentially higher priority levels. Continuity of the audio soundtrack playback is not assured using CODECs with synchronization method zero.

CODEC_SYNC_METHOD_DROP_FRAMES_IMMEDIATELY

The MMIO_DROP_DELTA_FRAME flag will be set whenever video is behind by more than a threshold. It is up to the CODEC implementation to determine the appropriate action to take when the MMIO_DROP_DELTA_FRAME flag is set. CODECs that are unable to drop individual delta frames within a delta frame sequence may choose to decompress the data but forgo display of the decompressed data to reduce processing complexity in an attempt to "catch up." Such an approach should be used, only to a certain extent, to ensure continuity of the audio soundtrack playback. That is, if the reduction in processing complexity is not sufficient to get caught up, and if drop frame requests continue to occur on consecutive frames, then the CODEC should further reduce its processing complexity by dropping frames completely until the next key frame.

CODEC_SYNC_METHOD_DROP_FRAMES_PRECEDING_KEY

The MMIO__DROP_DELTA_FRAME flag will be set whenever video is behind by more than a threshold, at points in the stream where the time remaining to the next key frame is less than or equal to the time interval by which the video is behind. In most cases, the effect of this method is that only delta frames preceding key frames are dropped. If the video falls behind and no key frames are encountered within two seconds in the data stream, delta frames will be dropped without regard to key frame proximity. If the video falls behind and the data stream contains consecutive key frames, key frames will be dropped.

CODECINIFILEINFO Field - ulReserved1

ulReserved1 (ULONG)

Reserved for future use and must be set to zero.

CODECINIFILEINFO Field - ulXalignment

ulXalignment (ULONG)

Indicates the top byte alignment of the starting address of the video window. This field is valid only when CODEC_MULAPERATURE and CODEC_DIRECT_DISPLAY are set. It is required for the system to place the top left corner of the video window on the specified boundary so the decompressor can improve performance. For example, 2 means the starting video memory address is on the word boundary. If 0 is specified, then 1-byte is assumed.

CODECINIFILEINFO Field - ulYalignment

ulYalignment (ULONG)

Indicates the top byte alignment of the starting address of the video window. This field is valid only when CODEC_MULAPERATURE and CODEC_DIRECT_DISPLAY are set. This information is provided by the block-oriented decompressor to allow the system to position the top left video window corner on a specific scan boundary to avoid dividing it between two 64KB apertures. The remainder of the 64 is divided by the *ulAlignment* and scale factor (for example, 2 for pel doubling) must be 0. This field is ignored when CODEC_SCALE_CONTINUOUS is specified.

CODECINIFILEINFO Field - ulSpecInfo[CODEC_INFO_SIZE]

ulSpecInfo[CODEC_INFO_SIZE] (ULONG)

Contains the CODEC-specific information.

CODECOPEN

This structure contains information that is used to open a CODEC. It describes the source and destination data format, as well as control and CODEC-specific information to indicate the operation of the CODEC. This structure is used on the [MMIOM_CODEC_OPEN](#) message.

```
typedef struct _CODECOPEN {
    ULONG    ulFlags;        /* Commands to CODEC Proc. */
    PVOID    pControlHdr;    /* CODEC control parameters. */
    PVOID    pSrcHdr;        /* Source header. */
    PVOID    pDstHdr;        /* Destination header. */
    PVOID    pOtherInfo;     /* Pointer to MMVIDEOPEN. */
} CODECOPEN;

typedef CODECOPEN *PCODECOPEN;
```

CODECOPEN Field - ulFlags

ulFlags (ULONG)

Contains commands to the CODEC Proc. Flags are the same as the *ulCapsFlags* field defined in the [CODECINIFILEINFO](#) structure. For example, if the flags `CODEC_8_BIT_COLOR` and `CODEC_DECOMPRESS` are specified, the returned instance handle will support decompression into 256 colors.

CODECOPEN Field - pControlHdr

pControlHdr (PVOID)

Contains the control parameters for compression or decompression. The file I/O procedure must propagate this control information to the CODEC Proc at CODEC initialization. The information is CODEC specific, for example, Huffman parameters. If NULL, the default is assumed.

CODECOPEN Field - pSrcHdr

pSrcHdr (PVOID)

Contains the source header for video compression or decompression. For video, it is [CODECVIDEOHEADER](#). See the [CODECVIDEOHEADER](#) structure for detailed information.

CODECOPEN Field - pDstHdr

pDstHdr (PVOID)

Contains the destination header for compression or decompression. For video, it is [CODECVIDEOHEADER](#).

Note: Fields not used in the structure must be set to NULL. See the [CODECVIDEOHEADER](#) structure for detailed information.

CODECOPEN Field - pOtherInfo

pOtherInfo ([PVOID](#))

For video, this points to the [MMVIDEOOPEN](#) structure. This field is only used if the CODEC has the CODEC_DATA_CONSTRAINT capability specified in the *ulCapsFlags* field of [CODECINIFILEINFO](#).

CODECVIDEOHEADER

This structure is the video CODEC header and it describes the data format of either the source video or the destination video. It is used on the [MMIOM_CODEC_OPEN](#) message.

```
typedef struct _CODECVIDEOHEADER {
    ULONG      ulStructLen;    /* Structure length. */
    ULONG      cx;             /* Width of bitmap. */
    ULONG      cy;             /* Height of bitmap. */
    USHORT     cPlanes;        /* Number of planes. */
    USHORT     cBitCount;       /* Bits per pel. */
    ULONG      ulColorEncoding; /* Color encoding. */
    GENPAL     genpal;          /* Generic palette. */
} CODECVIDEOHEADER;

typedef CODECVIDEOHEADER *PCODECVIDEOHEADER;
```

CODECVIDEOHEADER Field - ulStructLen

ulStructLen ([ULONG](#))

This field indicates the length of the CODECVIDEOHEADER structure.

CODECVIDEOHEADER Field - cx

cx ([ULONG](#))

Specifies the width of the bitmap, in pels.

CODECVIDEOHEADER Field - cy

cy ([ULONG](#))
Specifies the height of the bitmap, in pels.

CODECVIDEOHEADER Field - cPlanes

cPlanes ([USHORT](#))
Specifies the number of planes. The value is normally set to 1.

CODECVIDEOHEADER Field - cBitCount

cBitCount ([USHORT](#))
Specifies the number of bits per pel. Valid values are 1, 4, 8, 16, and 24.

CODECVIDEOHEADER Field - ulColorEncoding

ulColorEncoding ([ULONG](#))
Color encoding. If zero is specified the default is assumed.

MMIO_RGB_5_6_5
Each pel is an RGB_5_6_5 data type.

MMIO_RGB_24
Each pel is an RGB_24 data type.

MMIO_YUV_4_1_1
Each pel is a YUV_4_1_1 data type.

MMIO_YUV_24
Each pel is a YUV_24 data type.

MMIO_COMPRESSED
The data is compressed.

MMIO_PALETTIZED
The data is palettized. Number of bits per pel specified in *cBitCount*.

CODECVIDEOHEADER Field - genpal

genpal ([GENPAL](#))
For the *pSrcHdr* field of [CODECOPEN](#) it contains the video stream palette. For the *pDstHdr* field of [CODECOPEN](#) it contains the destination color space (for example, the physical palette if the CODEC_DIRECT_DISPLAY capability flag is set). The *ulStartIndex* field of the [GENPAL](#) structure is always NULL. The *prgb2Entries* field of the [GENPAL](#) structure is NULL when the number of colors

exceeds 256.

CONNECT

This structure is a data type used in the [MCI_SYSINFO_CONPARAMS](#) structure.

```
typedef struct _CONNECT {
    USHORT    usConnectType;           /* Connector type. */
    CHAR      szToInstallName[MAX_DEVICE_NAME]; /* Install name. */
    USHORT    usToConnectIndex;       /* Connector index. */
} CONNECT;

typedef CONNECT *PCONNECT;
```

CONNECT Field - usConnectType

usConnectType ([USHORT](#))
The connector type.

CONNECT Field - szToInstallName[MAX_DEVICE_NAME]

szToInstallName[MAX_DEVICE_NAME] ([CHAR](#))
The installation name this connector is connected to.

CONNECT Field - usToConnectIndex

usToConnectIndex ([USHORT](#))
The connector index to which this connector is connected.

CONTROL_PARM

This data structure contains fields for the *pParm* and *ulParmSize* fields of the [DDCMDCONTROL](#) data structure.

```
typedef struct _CONTROL_PARM {
    ULONG    ulTime; /* Time in milliseconds. */
} CONTROL_PARM;
```

CONTROL_PARM Field - ulTime

ulTime (ULONG)

Specifies the time in milliseconds. The stream handler sets the cue time when the *ulCmd* field of the [DDCMDCONTROL](#) data structure is DDCMD_ENABLE_EVENT. The PDD returns the current time for DDCMD_STOP and DDCMD_PAUSE.

DATA_EVCB

This structure describes a data cue-point event whose purpose is to set a cue point for a specific piece of data in the data stream. The stream handler reports a data cue-point event when the data is detected in the stream. This event can be enabled as a single or recurring event.

```
typedef struct _DATA_EVCB {
    ULONG      ulType;          /* Event type. */
    ULONG      ulSubType;       /* Event subtype. */
    ULONG      ulFlags;         /* Flags. */
    HSTREAM     hstream;        /* Stream handle. */
    HID         hid;            /* Stream handler ID. */
    ULONG      ulStatus;        /* Event status. */
    MMTIME      mmtimeStream;   /* Stream time. */
    ULONG      ulEventParm1;    /* Pointer to buffer. */
    ULONG      ulEventParm2;    /* Length of buffer. */
} DATA_EVCB;

typedef DATA_EVCB *PDATA_EVCB;
```

DATA_EVCB Field - ulType

ulType (ULONG)

Identifies the event type. For this event, the type must be EVENT_CUE_DATA.

DATA_EVCB Field - ulSubType

ulSubType (ULONG)

Identifies the data type to search for. Each stream handler defines a set of data types that it accepts.

DATA_EVCB Field - ulFlags

ulFlags (ULONG)
Flags.

- EVENT_SINGLE
Event reported on the first occurrence detected. This event remains enabled until disabled. If a stream seeks to a position before the cue point occurrence is performed, the cue point might be detected and reported again. This value is the default.
- EVENT_RECURRING
Event reported for each occurrence of the data in the data stream. This event remains enabled until disabled.
- EVENT_DATAPTR
Describes the use of the *ulEventParm1* and *ulEventParm2* fields.

DATA_EVCB Field - hstream

hstream (HSTREAM)
Stream handle that identifies the stream instance for this event. This field must be filled in by the application media control device for the [SpiEnableEvent](#) call when this event is enabled. It also must be filled in by the stream handler when reporting this event.

DATA_EVCB Field - hid

hid (HID)
Handler ID that identifies the stream handler that reported this event. This field can be filled in by the application media control device for the [SpiEnableEvent](#) call when this event is enabled. In this case, it identifies which stream handler must report the event. The stream handler must be able to report this event for the [SpiEnableEvent](#) call to succeed. This field also must be filled in by the stream handler when reporting this event.

DATA_EVCB Field - ulStatus

ulStatus (ULONG)
Event status.

DATA_EVCB Field - mmtimeStream

mmtimeStream (MMTIME)
Stream time at which the data cue point was detected.

DATA_EVCB Field - ulEventParm1

ulEventParm1 (ULONG)

If the EVENT_DATAPTR flag is set, this field is a pointer to a buffer that contains the data to search for in the data stream. Otherwise, this field represents the data to search for.

DATA_EVCB Field - ulEventParm2

ulEventParm2 (ULONG)

If the EVENT_DATAPTR flag is set, this field is the length of the buffer that contains the data to search for in the data stream. Otherwise, this field is not used.

DCB

This structure contains device specific-information. It is used at stream creation by the application media control device to deliver device-specific information to the source or target stream handlers.

```
typedef struct _DCB {
    ULONG    ulDCBLen;           /* Control block length. */
    SZ       szDevName[MAX_SPI_NAME]; /* Device driver name. */
} DCB;

typedef DCB FAR *PDCB;
```

DCB Field - ulDCBLen

ulDCBLen (ULONG)

Length of the device control block.

DCB Field - szDevName[MAX_SPI_NAME]

szDevName[MAX_SPI_NAME] (SZ)

Identifies the device driver that the stream handler connects to for a particular stream instance. For stream handler device drivers, inter-device driver communication (IDC) is used to call the physical device driver. The device driver name must not be more than eight characters (excluding the file extension).

DDCMDCOMMON

This data structure contains common fields between all DDCMD data structures.

```
typedef struct _ddcmd_common_parm {  
    ULONG      ulFunction; /* Function requested by stream handler. */  
    HSTREAM    hStream;    /* Data stream instance. */  
} DDCMDCOMMON;
```

DDCMDCOMMON Field - ulFunction

ulFunction (ULONG)

Specifies the function requested by the stream handler.

DDCMDCOMMON Field - hStream

hStream (HSTREAM)

Specifies the data stream instance for this event.

DDCMDCONTROL

This data structure contains fields for the [DDCMD_CONTROL](#) message.

```
typedef struct _ddcmd_control_parm {  
    ULONG      ulFunction; /* Function requested. */  
    HSTREAM    hStream;    /* Stream handle. */  
    HEVENT     hEvent;     /* Event handle. */  
    ULONG      ulCmd;       /* Flags. */  
    PVOID      pParm;       /* Pointer to CONTROL\_PARM. */  
    ULONG      ulParmSize; /* See CONTROL\_PARM. */  
} DDCMDCONTROL;
```

DDCMDCONTROL Field - ulFunction

ulFunction (ULONG)

Specifies the function requested by the stream handler.

DDCMDCONTROL Field - hStream

hStream ([HSTREAM](#))

Specifies the stream handle that identifies the stream instance for this event.

DDCMDCONTROL Field - hEvent

hEvent ([HEVENT](#))

Event handle.

DDCMDCONTROL Field - ulCmd

ulCmd ([ULONG](#))

Defines the following flags:

DDCMD_START

Starts a device.

DDCMD_STOP

Stops a device and returns the current stream time in milliseconds. All buffers in the PDD are flushed and everything is reset to 0. The device driver sends the current stream time back to the stream handler in milliseconds (using the *pParm* and *ulParmSize* fields). For example, return a ULONG of stream time (in milliseconds) and the ULONG of 4 bytes is input in the *ulParmSize* field.

DDCMD_PAUSE

Pauses the device and returns the current stream time in milliseconds. Unlike DDCMD_STOP, DDCMD_PAUSE is not destructive. The device driver is able to resume its function at a later time, exactly where it was paused. DDCMD_PAUSE returns the current stream time in milliseconds within the *pParm* field.

DDCMD_RESUME

Resumes a currently paused stream. DDCMD_RESUME is only honored if the stream was paused previously. If the stream was stopped and the stream handler issues DDCMD_RESUME, an ERROR_INVALID_SEQUENCE is returned.

DDCMD_ENABLE_EVENT

Enables event detection if your PDD supports events. The event message is sent to the device driver and reviews the structure. The *hEvent* field is a unique handle that signifies the event. *pParm* contains the stream time with which the event should be detected. This command would be honored whether currently in a streaming state, pause, or stop.

For example, a stream handler is programmed to be detected at 1000 milliseconds in the stream. It sends a DDCMD_ENABLE_EVENT to the device driver with a unique handle event (*hEvent*) and 1000 milliseconds stated in the *pParm* field. The device driver obtains the handle event, makes sure there are no duplicate entries, and adds the *hEvent* to a link list or array (depending on how the device driver is implemented). The device driver now knows it must report an event back to the stream handler when 1000 milliseconds has been reached in the stream. See [SHD_REPORT_EVENT](#) for further information.

DDCMD_DISABLE_EVENT

Disables event detection in the device driver. An event handle is passed from the structure. The device driver searches its linked list or array to find the matching event handle and deletes it from the list.

DDCMD_PAUSE_TIME

Pauses the time but not the stream.

DDCMD_RESUME_TIME
Resumes the time.

DDCMDCONTROL Field - pParm

pParm ([PVOID](#))
Pointer to the [CONTROL_PARM](#) data structure.

DDCMDCONTROL Field - ulParmSize

ulParmSize ([ULONG](#))
Size of the [CONTROL_PARM](#) data structure.

DDCMDDEREGISTER

This data structure contains fields for the [DDCMD_DEREG_STREAM](#) message.

```
typedef struct _ddcmd_deregister_parm {  
    ULONG         ulFunction; /* Function requested by stream handler. */  
    HSTREAM       hStream;    /* Stream handle. */  
} DDCMDDEREGISTER;
```

DDCMDDEREGISTER Field - ulFunction

ulFunction ([ULONG](#))
Specifies the function requested by the stream handler.

DDCMDDEREGISTER Field - hStream

hStream ([HSTREAM](#))
Specifies the stream handle needed at interrupt time.

DDCMDREADWRITE

This data structure contains fields for the [DDCMD_READ](#) and [DDCMD_WRITE](#) messages.

```
typedef struct _ddcmd_readwrite_parm {
    ULONG      ulFunction;    /* Function requested by stream handler. */
    HSTREAM     hStream;      /* Stream handle. */
    PVOID       pBuffer;      /* Buffer pointer. */
    ULONG      ulBufferSize;  /* Buffer size. */
    PVOID       pProcessLin;   /* Process linear record pointer. */
    BOOL        fEOS;         /* Whether or not this is the EOS buffer. */
    ULONG      ulParm1;        /* Reserved for future use. */
    ULONG      ulParm2;        /* Reserved for future use. */
    ULONG      ulLength;       /* Length of the structure. */
} DDCMDREADWRITE;
```

DDCMDREADWRITE Field - ulFunction

ulFunction ([ULONG](#))

Specifies the function requested by the stream handler.

DDCMDREADWRITE Field - hStream

hStream ([HSTREAM](#))

Specifies the stream handle that identifies the stream instance for this event.

DDCMDREADWRITE Field - pBuffer

pBuffer ([PVOID](#))

Buffer pointer of the empty buffer that the device driver fills from its record operation.

DDCMDREADWRITE Field - ulBufferSize

ulBufferSize ([ULONG](#))

Specifies the buffer size of the empty buffer.

DDCMDREADWRITE Field - pProcessLin

pProcessLin ([PVOID](#))

Provides the 32-bit process linear address to a Ring 3 stream handler so the stream handler will have access to the buffers if a GDT or physical address is requested.

DDCMDREADWRITE Field - fEOS

fEOS ([BOOL](#))

Whether or not this is the EOS buffer.

DDCMDREADWRITE Field - ulParm1

ulParm1 ([ULONG](#))

For MPEG, contains the initial audio system clock reference (SCR).

DDCMDREADWRITE Field - ulParm2

ulParm2 ([ULONG](#))

For MPEG, contains the Presentation Time Stamp (PTS).

DDCMDREADWRITE Field - ulLength

ulLength ([ULONG](#))

Length of the structure.

DDCMDREGISTER

This data structure contains fields for the [DDCMD_REG_STREAM](#) message.

```
typedef struct _ddcmd_register_parm {  
    ULONG      ulFunction;          /* Function requested by stream handler. */  
};
```

```

HSTREAM    hStream;          /* Stream handle. */
ULONG      ulSysFileNum;     /* Device handle. */
PSHDFN     pSHDEntryPoint;   /* Stream handler entry point. */
ULONG      ulStreamOperation; /* Stream operation. */
SPCBKEY     spcbkey;         /* Input to the VSD or PDD (optional) buffer size. */
ULONG      ulBufSize;        /* VSD or PDD output buffer size in bytes. */
ULONG      ulNumBufs;        /* VSD or PDD output total number of buffers for SPCB. */
ULONG      ulAddressType;     /* VSD or PDD output (required) addr ptr type to data buffer. */
ULONG      ulBytesPerUnit;    /* VSD or PDD output (unused, set to zero). */
MMTIME     mmtimePerUnit;    /* VSD or PDD output (Unused, set to zero). */
E_DCB      dcbAudio;         /* Stream handler DCB. */
HID        hid;              /* Stream handler ID. */
} DDCMDREGISTER;

```

DDCMDREGISTER Field - ulFunction

ulFunction ([ULONG](#))

Specifies the function requested by the stream handler.

DDCMDREGISTER Field - hStream

hStream ([HSTREAM](#))

Specifies the stream handle to communicate with the stream handler at interrupt time.

DDCMDREGISTER Field - ulSysFileNum

ulSysFileNum ([ULONG](#))

Specifies the device handle so the VSD or PDD can map the device instance to *hStream*. This field has to map with how the device has been initialized during AUDIO_INIT.

DDCMDREGISTER Field - pSHDEntryPoint

pSHDEntryPoint ([PSHDFN](#))

Specifies the stream handler entry point so that the device driver can call back into the stream handler during streaming time to report interrupts and events. For VSDs this is a user-level stream handler entry point.

DDCMDREGISTER Field - ulStreamOperation

ulStreamOperation (ULONG)

Specifies the stream operation (record or playback). The device driver should verify with how it has been initialized through IOctls. The following stream operations are defined (VSDs should verify with VSD_SET or VSD_OPEN settings):

- STREAM_OPERATION_CONSUME
Playback.
- STREAM_OPERATION_PRODUCE
Record.

DDCMDREGISTER Field - spcbkey

spcbkey (SPCBKEY)

Specifies input to the VSD or PDD (optional) buffer size.

DDCMDREGISTER Field - ulBufSize

ulBufSize (ULONG)

Specifies VSD or PDD output buffer size in bytes that will be sent to the driver. The caller will fill in a default value. If the device driver does not override this value, it will be used to determine the buffer size.

Note: The device driver sets this field to indicate the buffer size that it would like to receive from the streaming subsystem. For MIDI, this field is usually set to 512. For DMA-based audio or other systems where the interrupt rate equals the buffer rate, this field should be set to about 1/30 the quantity of data consumed or produced per second. This field supplies the system with a minimal buffer size for wave audio. Because other system elements also negotiate to determine buffer size and for software motion video, the system might provide buffers smaller than the size requested.

DDCMDREGISTER Field - ulNumBufs

ulNumBufs (ULONG)

Specifies VSD or PDD output total number of buffers that will be allocated. The caller will fill in a default value. If the device driver does not override this value, it will be used to determine the total number of buffers.

DDCMDREGISTER Field - ulAddressType

ulAddressType (ULONG)

Specifies VSD or PDD output (required) address pointer type to data buffer. The VSD or PDD tells the stream handler what type of address pointer it expects the data buffer to be. The stream handler then requests this address type to the SSM, so that the Sync/Stream Manager will send the correct type for each buffer request.

- ADDRESS_TYPE_VIRTUAL
0

ADDRESS_TYPE_PHYSICAL
1
ADDRESS_TYPE_LINEAR
2

DDCMDREGISTER Field - ulBytesPerUnit

ulBytesPerUnit ([ULONG](#))

Specifies VSD or PDD output (unused, set to zero). Device driver timing.

DDCMDREGISTER Field - mmtimePerUnit

mmtimePerUnit ([MMTIME](#))

Specifies VSD or PDD output. Device driver timing. (Unused, set to zero.)

DDCMDREGISTER Field - dcbAudio

dcbAudio ([E_DCB](#))

Specifies the input stream handler device control block (DCB).

DDCMDREGISTER Field - hid

hid ([HID](#))

Specifies the input stream handler ID to be used for all communication with the stream handler.

DDCMDSETUP

This data structure contains fields for the [DDCMD_SETUP](#) message.

```
typedef struct _ddcmd_setup_parm {  
    ULONG          ulFunction;      /* Function requested by stream handler. */  
    HSTREAM        hStream;         /* Stream handle. */  
    PSETUP_PARM    pSetupParm;     /* Pointer to SETUP_PARM. */  
    ULONG          ulSetupParmSize; /* Size of SETUP_PARM. */  
} DDCMDSETUP;
```

DDCMDSETUP Field - ulFunction

ulFunction ([ULONG](#))

Specifies the function requested by the stream handler.

DDCMDSETUP Field - hStream

hStream ([HSTREAM](#))

Specifies the stream handle that identifies the stream instance for this event.

DDCMDSETUP Field - pSetupParm

pSetupParm ([PSETUP_PARM](#))

Points to the [SETUP_PARM](#) data structure.

DDCMDSETUP Field - ulSetupParmSize

ulSetupParmSize ([ULONG](#))

The size of the [SETUP_PARM](#) data structure.

DDCMDSTATUS

This data structure contains fields for the [DDCMD_STATUS](#) message.

```
typedef struct _ddcmd_status_parm {  
    ULONG          ulFunction;    /* Function requested by stream handler. */  
    HSTREAM        hStream;      /* Stream handle. */  
    PSTATUS\_PARM   pStatus;      /* Pointer to STATUS_PARM. */  
    ULONG          ulStatusSize; /* Size or position time. */  
} DDCMDSTATUS;
```

DDCMDSTATUS Field - ulFunction

ulFunction ([ULONG](#))

Specifies the function requested by the stream handler.

DDCMDSTATUS Field - hStream

hStream ([HSTREAM](#))

Specifies the stream handle that identifies the stream instance for this event.

DDCMDSTATUS Field - pStatus

pStatus ([PSTATUS_PARM](#))

A pointer to the current position time as output. See [STATUS_PARM](#).

DDCMDSTATUS Field - ulStatusSize

ulStatusSize ([ULONG](#))

Specifies the size or position time as output.

DLGTEMPLATE

Dialog-template structure.

```
typedef struct _DLGTEMPLATE {
    USHORT    cbTemplate;    /* Length of template. */
    USHORT    type;          /* Template format type. */
    USHORT    codepage;      /* Code page. */
    USHORT    offadlgti;     /* Offset to dialog items. */
    USHORT    fsTemplateStatus; /* Template status. */
    USHORT    iItemFocus;    /* Index of item to receive focus initially. */
    USHORT    coffPresParams; /* Count of presentation-parameter offsets. */
    DLGITEM   adlgti[1];     /* Start of dialog items. */
} DLGTEMPLATE;

typedef DLGTEMPLATE *PDLGTEMPLATE;
```

DLGTEMPLATE Field - cbTemplate

cbTemplate (USHORT)
Length of template.

DLGTEMPLATE Field - type

type (USHORT)
Template format type.

DLGTEMPLATE Field - codepage

codepage (USHORT)
Code page.

DLGTEMPLATE Field - offadlgti

offadlgti (USHORT)
Offset to dialog items.

DLGTEMPLATE Field - fsTemplateStatus

fsTemplateStatus (USHORT)
Template status.

DLGTEMPLATE Field - itemFocus

itemFocus (USHORT)
Index of item to receive focus initially.

DLGTEMPLATE Field - coffPresParams

coffPresParams (USHORT)

Count of presentation-parameter offsets.

DLGTEMPLATE Field - adlgti[1]

adlgti[1] (DLGTITEM)

Start of dialog items.

DLGTITEM

Dialog-item structure.

```
typedef struct _DLGTITEM {
    USHORT    fsItemStatus;    /* Status. */
    USHORT    cChildren;       /* Count of children to this dialog item. */
    USHORT    cchClassLen;     /* Length of class name. */
    USHORT    offClassName;    /* Offset to class name. */
    USHORT    cchTextLen;      /* Length of text. */
    USHORT    offText;         /* Offset to text. */
    ULONG     flStyle;          /* Dialog item window style. */
    SHORT     x;               /* X-coordinate of origin of dialog-item window. */
    SHORT     y;               /* Y-coordinate of origin of dialog-item window. */
    SHORT     cx;              /* Dialog-item window width. */
    SHORT     cy;              /* Dialog-item window height. */
    USHORT    id;              /* Identity. */
    USHORT    offPresParams;    /* Reserved. */
    USHORT    offCtlData;      /* Offset to control data. */
} DLGTITEM;

typedef DLGTITEM *PDLGTITEM;
```

DLGTITEM Field - fsItemStatus

fsItemStatus (USHORT)

Status.

DLGTITEM Field - cChildren

cChildren (USHORT)
Count of children to this dialog item.

DLGTITEM Field - cchClassLen

cchClassLen (USHORT)
If zero, *offClassName* contains the hexadecimal equivalent of a preregistered class name.

DLGTITEM Field - offClassName

offClassName (USHORT)
If *cchClassLen* is nonzero, this is the offset to a null-terminated ASCII string that contains the classname. If *cchClassLen* is zero, this is of the form 0xhhh, where hhhh is the hexadecimal equivalent of the preregistered class name.

DLGTITEM Field - cchTextLen

cchTextLen (USHORT)
Length of text.

DLGTITEM Field - offText

offText (USHORT)
Offset to text.

DLGTITEM Field - flStyle

flStyle (ULONG)
The high-order 16 bits are the standard WS_* style bits. The low-order 16 bits are available for class-specific use.

DLGTITEM Field - x

x ([SHORT](#))
X-coordinate of origin of dialog-item window.

DLGTITEM Field - y

y ([SHORT](#))
Y-coordinate of origin of dialog-item window.

DLGTITEM Field - cx

cx ([SHORT](#))
Dialog-item window width.

DLGTITEM Field - cy

cy ([SHORT](#))
Dialog-item window height.

DLGTITEM Field - id

id ([USHORT](#))
Identity.

DLGTITEM Field - offPresParams

offPresParams ([USHORT](#))
Reserved.

DLGTITEM Field - offCtlData

offCtlData ([USHORT](#))
Offset to control data.

DIVE_CAPS

This structure contains fields for the [DiveQueryCaps](#) function.

```
typedef struct _DIVE_CAPS {  
    ULONG      ulStructLen;           /* Length of structure. */  
    ULONG      ulPlaneCount;         /* Number of defined planes. */  
    BOOL       fScreenDirect;        /* Direct screen access. */  
    BOOL       fBankSwitched;        /* Direct access requires bank-switch. */  
    ULONG      ulDepth;              /* Number of bits per pel. */  
    ULONG      ulHorizontalResolution; /* Screen width in pels. */  
    ULONG      ulVerticalResolution; /* Screen height in pels. */  
    ULONG      ulScanLineBytes;      /* Screen scan line (in bytes). */  
    FOURCC     fccColorEncoding;     /* Color encoding of the screen. */  
    ULONG      ulApertureSize;        /* Size of VRAM aperture in bytes. */  
    ULONG      ulInputFormats;        /* Number of input color formats. */  
    ULONG      ulOutputFormats;       /* Number of output color formats. */  
    ULONG      ulFormatLength;        /* Length of format buffer. */  
    PVOID      pFormatData;          /* Pointer to color-format buffer FOURCCs. */  
} DIVE_CAPS;  
  
typedef DIVE_CAPS *PDIVE_CAPS;
```

DIVE_CAPS Field - ulStructLen

ulStructLen ([ULONG](#))
Length of structure.

DIVE_CAPS Field - ulPlaneCount

ulPlaneCount ([ULONG](#))
Indicates number of planes available on the hardware device.

DIVE_CAPS Field - fScreenDirect

fScreenDirect ([BOOL](#))
Direct screen access. TRUE if addressability to VRAM is possible.

DIVE_CAPS Field - fBankSwitched

fBankSwitched (BOOL)

TRUE if VRAM is bank-switched. Direct access requires bank-switch.

DIVE_CAPS Field - ulDepth

ulDepth (ULONG)

Number of bits per pel.

DIVE_CAPS Field - ulHorizontalResolution

ulHorizontalResolution (ULONG)

Screen width in pels.

DIVE_CAPS Field - ulVerticalResolution

ulVerticalResolution (ULONG)

Screen height in pels.

DIVE_CAPS Field - ulScanLineBytes

ulScanLineBytes (ULONG)

Screen scan line (in bytes).

DIVE_CAPS Field - fccColorEncoding

fccColorEncoding (FOURCC)

Color encoding of the screen.

DIVE_CAPS Field - ulApertureSize

ulApertureSize (ULONG)
Size of VRAM aperture in bytes.

DIVE_CAPS Field - ullInputFormats

ullInputFormats (ULONG)
Number of input color formats.

DIVE_CAPS Field - ulOutputFormats

ulOutputFormats (ULONG)
Number of output color formats.

DIVE_CAPS Field - ulFormatLength

ulFormatLength (ULONG)
Length of format buffer.

DIVE_CAPS Field - pFormatData

pFormatData (PVOID)
Pointer to color-format buffer FOURCCs.

E_DCB

This structure contains device-specific information. It is used at stream creation by the application media control device to deliver device-specific information to the source or target stream handlers.

```
typedef struct _E_DCB {
```

```

    ULONG    ulDCBLen;           /* Length of structure. */
    SZ        szDevName[MAX_SPI_NAME]; /* Device driver name. */
    ULONG    ulSysFileNum;       /* File handle number. */
} E_DCB;

typedef E_DCB FAR *PE_DCB;

```

E_DCB Field - ulDCBLen

ulDCBLen ([ULONG](#))
Length of the device control block.

E_DCB Field - szDevName[MAX_SPI_NAME]

szDevName[MAX_SPI_NAME] ([SZ](#))
Identifies the device driver that the stream handler connects to for a particular stream instance. For stream handler device drivers, inter-device communication (IDC) is used to call the physical device driver. The device driver name must not be more than eight characters (excluding the file extension).

E_DCB Field - ulSysFileNum

ulSysFileNum ([ULONG](#))
File handle number.

EPSRCBUFTAB

This structure is the source buffer table for the [PARM_NOTIFY](#) data structure.

```

typedef struct _EPSRCBUFTAB {
    PVOID    pBuffer;           /* Pointer to buffer. */
    PVOID    pRecord;           /* Pointer to record in buffer. */
    ULONG    ulLength;          /* Buffer length. */
    ULONG    ulMessageParm;     /* Message to passed to application. */
    MMTIME   mmtimeOffset;      /* MMTIME offset from beginning of buffer. */
    ULONG    ulParm1;           /* Record or buffer-specific data. */
    ULONG    ulParm2;           /* Record or buffer-specific data. */
    PVOID    pProcessLin;       /* Process linear record pointer. */
} EPSRCBUFTAB;

typedef EPSRCBUFTAB FAR *PEPSRCBUFTAB;

```

EPSRCBUFTAB Field - pBuffer

pBuffer (PVOID)
Pointer to buffer.

EPSRCBUFTAB Field - pRecord

pRecord (PVOID)
This pointer is for split streams only.

EPSRCBUFTAB Field - ulLength

ulLength (ULONG)
Maximum buffer length on GetEmpty, Filled (actual) record/buffer length on ReturnFull.

EPSRCBUFTAB Field - ulMessageParm

ulMessageParm (ULONG)
Message to passed to application.

EPSRCBUFTAB Field - mmtimeOffset

mmtimeOffset (MMTIME)
MMTIME offset from beginning of buffer.

EPSRCBUFTAB Field - ulParm1

ulParm1 (ULONG)
Record or buffer-specific data.

EPSRCBUFTAB Field - ulParm2

ulParm2 ([ULONG](#))

Record or buffer-specific data.

EPSRCBUFTAB Field - pProcessLin

pProcessLin ([PVOID](#))

Provides the 32-bit process linear address to a Ring 3 stream handler so the stream handler will have access to the buffers if a GDT or physical address is requested.

EPTGTBUFTAB

This structure is the target buffer table for the [PARM_NOTIFY](#) data structure.

```
typedef struct _EPTGTBUFTAB {
    PVOID      pBuffer;      /* Pointer to buffer. */
    ULONG      ulBufId;      /* Buffer ID. */
    ULONG      ulLength;     /* Buffer length. */
    ULONG      ulMessageParm; /* Message passed to application. */
    MMTIME     mmtimeOffset; /* MMTIME offset from beginning of buffer. */
    ULONG      ulParm1;      /* Buffer-specific data. */
    ULONG      ulParm2;      /* Buffer-specific data. */
    PVOID      pProcessLin;  /* Process linear buffer pointer. */
} EPTGTBUFTAB;
```

```
typedef EPTGTBUFTAB FAR *PEPTGTBUFTAB;
```

EPTGTBUFTAB Field - pBuffer

pBuffer ([PVOID](#))

Pointer to buffer.

EPTGTBUFTAB Field - ulBufId

ulBufId ([ULONG](#))

The buffer ID is passed to the stream handler on GetFull. It must be passed back to the SSM on ReturnEmpty.

EPTGTBUFTAB Field - ulLength

ulLength ([ULONG](#))

Filled (actual) buffer length on GetFull, unused on ReturnEmpty.

EPTGTBUFTAB Field - ulMessageParm

ulMessageParm ([ULONG](#))

Message passed to application.

EPTGTBUFTAB Field - mmtimeOffset

mmtimeOffset ([MMTIME](#))

MMTIME offset from beginning of buffer.

EPTGTBUFTAB Field - ulParm1

ulParm1 ([ULONG](#))

Buffer-specific data.

EPTGTBUFTAB Field - ulParm2

ulParm2 ([ULONG](#))

Buffer-specific data.

EPTGTBUFTAB Field - pProcessLin

pProcessLin ([PVOID](#))

Provides the 32-bit process linear address to a Ring 3 stream handler so the stream handler will have access to the buffers if a GDT or physical address is requested.

ESRCBUFTAB

This structure is the source buffer table for the [PARM_NOTIFY](#) data structure.

```
typedef struct _ESRCBUFTAB {
    PVOID      pBuffer;          /* Pointer to buffer. */
    PVOID      pRecord;          /* Pointer to record in buffer. */
    ULONG      ulLength;         /* Buffer length. */
    ULONG      ulMessageParm;     /* Message to passed to application. */
    MMTIME     mmtimeOffset;     /* MMTIME offset from beginning of buffer. */
    ULONG      ulParm1;          /* Record or buffer specific data. */
    ULONG      ulParm2;          /* Record or buffer specific data. */
} ESRCBUFTAB;

typedef ESRCBUFTAB FAR *PESRCBUFTAB;
```

ESRCBUFTAB Field - pBuffer

pBuffer ([PVOID](#))
Pointer to buffer.

ESRCBUFTAB Field - pRecord

pRecord ([PVOID](#))
This pointer is for split streams only.

ESRCBUFTAB Field - ulLength

ulLength ([ULONG](#))
Maximum buffer length on GetEmpty, Filled (actual) record/buffer length on ReturnFull.

ESRCBUFTAB Field - ulMessageParm

ulMessageParm ([ULONG](#))
Message to passed to application.

ESRCBUFTAB Field - mmtimeOffset

mmtimeOffset ([MMTIME](#))
MMTIME offset from beginning of buffer.

ESRCBUFTAB Field - ulParm1

ulParm1 ([ULONG](#))
Record or buffer specific data.

ESRCBUFTAB Field - ulParm2

ulParm2 ([ULONG](#))
Record or buffer specific data.

ETGTBUFTAB

This structure is the target buffer table for the [PARM_NOTIFY](#) data structure.

```
typedef struct _ETGTBUFTAB {  
    PVOID      pBuffer;      /* Pointer to buffer. */  
    ULONG      ulBufId;      /* Buffer ID. */  
    ULONG      ulLength;     /* Buffer length. */  
    ULONG      ulMessageParm; /* Message passed to application. */  
    MMTIME     mmtimeOffset; /* MMTIME offset from beginning of buffer. */  
    ULONG      ulParm1;      /* Buffer-specific data. */  
    ULONG      ulParm2;      /* Buffer-specific data. */  
} ETGTBUFTAB;  
  
typedef ETGTBUFTAB FAR *PETGTBUFTAB;
```

ETGTBUFTAB Field - pBuffer

pBuffer ([PVOID](#))
Pointer to buffer.

ETGTBUFTAB Field - ulBufId

ulBufId (ULONG)
The buffer id is passed to the stream handler on GetFull. It must be passed back to the SSM on ReturnEmpty.

ETGTBUFTAB Field - ulLength

ulLength (ULONG)
Filled (actual) buffer length on GetFull, unused on ReturnEmpty.

ETGTBUFTAB Field - ulMessageParm

ulMessageParm (ULONG)
Message passed to application.

ETGTBUFTAB Field - mmtimeOffset

mmtimeOffset (MMTIME)
MMTIME offset from beginning of buffer.

ETGTBUFTAB Field - ulParm1

ulParm1 (ULONG)
Buffer-specific data.

ETGTBUFTAB Field - ulParm2

ulParm2 (ULONG)
Buffer-specific data.

EVCB

This structure describes an event. It is used by an application media control device to enable an event for an [SpiEnableEvent](#) call. Stream handlers use this same data structure to report events. Most of the system-defined events have a corresponding event control block, which is the same structure with some different field names and uses.

```
typedef struct _EVCB {
    ULONG      ulType;           /* Event type. */
    ULONG      ulSubType;        /* Event subtype. */
    ULONG      ulFlags;          /* Flags. */
    HSTREAM     hstream;         /* Stream handle. */
    HID         hid;             /* Stream handler ID. */
    ULONG      ulStatus;         /* Event status. */
    ULONG      ulEventParm1;     /* I/O parameter. */
    ULONG      ulEventParm2;     /* I/O parameter. */
    ULONG      ulEventParm3;     /* I/O parameter. */
} EVCB;

typedef EVCB FAR *PEVCB;
```

EVCB Field - ulType

ulType ([ULONG](#))

Identifies the event type. There are several system-defined events available. See the specific event control blocks for these events.

EVCB Field - ulSubType

ulSubType ([ULONG](#))

Identifies the event subtype. This field is used to further subdivide an event type.

EVCB Field - ulFlags

ulFlags ([ULONG](#))

Set of bit flags with various meanings.

EVCB Field - hstream

hstream ([HSTREAM](#))

Stream handle that identifies the stream instance for this event. This field must be filled in by the application media control device for the [SpiEnableEvent](#) call when this event is enabled. It also must be filled in by the stream handler when reporting this event.

EVCB Field - hid

hid ([HID](#))
ID that identifies the stream handler that reported this event. This field may be filled in by the application media control device for the [SpiEnableEvent](#) call when this event is enabled. In this case, it identifies which stream handler must report this event. The stream handler must be able to report this event for the [SpiEnableEvent](#) call to succeed. This field also must be filled in the stream handler when reporting this event.

EVCB Field - ulStatus

ulStatus ([ULONG](#))
Status of the event. For error events, this field usually contains the error return code.

EVCB Field - ulEventParm1

ulEventParm1 ([ULONG](#))
Input or output parameter. This is defined by the stream handler for each event type.

EVCB Field - ulEventParm2

ulEventParm2 ([ULONG](#))
Input or output parameter. This is defined by the stream handler for each event type.

EVCB Field - ulEventParm3

ulEventParm3 ([ULONG](#))
Input or output parameter. This is defined by the stream handler for each event type.

FOURCC

This data type contains a four-character code that is used as an identifier for I/O procedures and CODECs.

```
typedef ULONG FOURCC;
```

GBTNCDATA

The data structure contains information used as the graphic button control data.

```
typedef struct _GBTNCDATA {
    USHORT    usReserved;    /* Reserved field. */
    PSZ       pszText;       /* Initial text of button. */
    HMODULE    hmod;         /* Handle of bitmap resource. */
    USHORT    cBitmaps;      /* Number of bitmaps. */
    USHORT    aidBitmap[1];  /* Array of bitmap IDs. */
} GBTNCDATA;

typedef GBTNCDATA *PGBTNCDATA;
```

GBTNCDATA Field - usReserved

usReserved ([USHORT](#))

Constant that indicates that the structure comes from a resource. This field must be set to GB_RESOURCE.

GBTNCDATA Field - pszText

pszText ([PSZ](#))

This character-string field provides the initial text for the button. Mnemonics are supported and, like push buttons, are specified by the tilde (~) character. If no text is wanted, a NULL string, "", must be specified; this is a variable-length character string that is null terminated. The *cBitmaps* field follows the null terminator.

GBTNCDATA Field - hmod

hmod ([HMODULE](#))

Handle of bitmap resource.

GBTNCDATA Field - cBitmaps

cBitmaps (USHORT)

This field indicates the number of bitmaps associated with the graphic button. This number does not necessarily mean unique bitmaps. Duplicate bitmaps can exist in the graphic button's collection for animation purposes.

GBTNCDATA Field - aidBitmap[1]

aidBitmap[1] (USHORT)

This field provides the array of bitmap resource IDs. Each bitmap is assigned an index (0-based).

GENPAL

This structure is a generic palette and it defines a color palette of up to 256 entries. This structure is used as part of the CODECVIDEOHEADER structure.

```
typedef struct _GENPAL {
    ULONG      ulStartIndex; /* Starting RGB index. */
    ULONG      ulNumColors;  /* Number of following entries. */
    PRGB2      prgb2Entries; /* 256 RGB entries. */
} GENPAL;

typedef GENPAL *PGENPAL;
```

GENPAL Field - ulStartIndex

ulStartIndex (ULONG)

Starting RGB index.

GENPAL Field - ulNumColors

ulNumColors (ULONG)

Number of following entries.

GENPAL Field - prgb2Entries

prgb2Entries ([PRGB2](#))
256 RGB entries.

HAND

This structure contains a stream handler name and its class. It is used in the [SpiEnumerateHandlers](#) call.

```
typedef struct _HAND {  
    SZ      szHandlerClass[MAX_SPI_NAME]; /* Stream handler class. */  
    SZ      HandlerName[MAX_SPI_NAME];    /* Stream handler name. */  
} HAND;  
  
typedef HAND *PHAND;
```

HAND Field - szHandlerClass[MAX_SPI_NAME]

szHandlerClass[MAX_SPI_NAME] ([SZ](#))

Stream handler class name. Stream handlers are grouped into the following supported classes:

- AUDIOSYS
 - FILESYS
 - MEMSYS
 - MIDISYS
 - NULLSYS
 - VIDEOSYS
-

HAND Field - HandlerName[MAX_SPI_NAME]

HandlerName[MAX_SPI_NAME] ([SZ](#))

Stream handler name.

HBITMAP

Bit-map handle.

```
typedef LHANDLE HBITMAP;
```

HCODEC

A CODEC instance handle returned by the [MMIOM_CODEC_OPEN](#) message. It is used on other CODEC messages to identify the instance.

```
typedef ULONG HCODEC;
```

HDC

Device-context handle.

```
typedef LHANDLE HDC;
```

HDIVE

DIVE instance.

```
typedef ULONG HDIVE;
```

HEVENT

Event handle.

```
typedef ULONG HEVENT;
```

HID

Stream handler ID.

```
typedef ULONG HID;
```

HMMCF

A RIFF compound file instance handle returned by the [mmioCFOpen](#) function. It is used with the other compound file functions to identify the instance.

```
typedef HMMIO HMMCF;
```

HMMIO

An MMIO data object instance handle returned by the [mmioOpen](#) function. It is used with other data object functions and messages to identify the instance.

```
typedef ULONG HMMIO;
```

HMODULE

Module handle.

```
typedef LHANDLE HMODULE;
```

HPOINTER

Mouse-pointer handle.

```
typedef LHANDLE HPOINTER;
```

HQUEUE

Value (32-bit) used as a system queue handle.

```
typedef LHANDLE HQUEUE;
```

HRGN

Region handle.

```
typedef LHANDLE HRGN;
```

HSTREAM

Stream handle.

```
typedef ULONG HSTREAM;
```

HVSD

VSD handle.

```
typedef PVOID HVSD;
```

HWND

Window handle.

```
typedef LHANDLE HWND;
```

IMPL_EVCB

This structure describes an implicit event. A pointer to this data structure is passed in an [SpiCreateStream](#) call. It must be initialized to 0 for this call. Implicit events are reported using this data structure. The application media control device must support notification of these events. These events are enabled implicitly at stream creation and cannot be disabled.

```
typedef struct _IMPL_EVCB {
    ULONG      ulType;      /* Event type. */
    ULONG      ulSubType;   /* Event subtype. */
    ULONG      ulFlags;     /* Flags. */
    HSTREAM     hStream;    /* Stream handle. */
    HID         hid;        /* Stream handler ID. */
    ULONG      ulStatus;    /* Event status. */
    ULONG      unused1;     /* Not used. */
    ULONG      unused2;     /* Not used. */
    ULONG      unused3;     /* Not used. */
} IMPL_EVCB;

typedef IMPL_EVCB FAR *PIMPL_EVCB;
```

IMPL_EVCB Field - ulType

ulType (ULONG)

Identifies the event type. For this event, the type must be EVENT_IMPLICIT_TYPE.

IMPL_EVCB Field - ulSubType

ulSubType (ULONG)

Identifies the event subtype. This field is used to further subdivide an event type. (See the stream handler module section for more details.) Following is a list of system-defined implicit events:

EVENT_EOS

End of stream event. This event is generated after the target stream handler has consumed the last buffer in the stream. This event notifies the application media control device that the stream has completed.

EVENT_ERROR

An error has occurred in a stream handler or device driver while streaming. The *ulStatus* field contains the error return code. Some stream handlers also set some flags in the *ulFlags* field to further describe the severity of the error.

EVENT_PLAYLISTCUEPOINT

A memory stream handler playlist cue-point event, used when streaming data to or from the memory stream handler. See the playlist-specific event control block ([PLAYL_EVCB](#)) for the EVCB field definitions.

EVENT_PLAYLISTMESSAGE

A memory stream handler playlist message, used when streaming data to or from the memory stream handler. See the playlist-specific event control block ([PLAYL_EVCB](#)) for the EVCB field definitions.

EVENT_QUEUE_OVERFLOW

Event queue overflow. Indicates that an event has been lost due to too many events being generated. The application media control device must use this event to clear any waiting conditions. See *Performance-Tuning the Stream Manager* in the *OS/2 Multimedia Subsystem Programming Guide* for more information on increasing the event queue size.

EVENT_STREAM_STOPPED

Asynchronous notification of the completion of an [SpiStopStream](#) call. Both the flush stop and discard stop are asynchronous, while the pause stop is synchronous. See the [SpiStopStream](#) function.

EVENT_SYNC_PREROLLED

Asynchronous notification of the completion of an [SpiStartStream](#) call. The preroll start is asynchronous, while the start stream is synchronous. All synchronized streams are prerolled. See the [SpiStartStream](#) function.

IMPL_EVCB Field - ulFlags

ulFlags (ULONG)

Set of bit flags with various meanings.

IMPL_EVCB Field - hStream

hStream (HSTREAM)

Stream handle that identifies the stream instance for this event. This field must be filled in by the stream handler when reporting this event.

IMPL_EVCB Field - hid

hid ([HID](#))
Handler ID that identifies the stream handler that reported this event. This field must be filled in by the stream handler when reporting this event.

IMPL_EVCB Field - ulStatus

ulStatus ([ULONG](#))
Status of the event. For error events, this field contains the error return code.

IMPL_EVCB Field - unused1

unused1 ([ULONG](#))
Not used.

IMPL_EVCB Field - unused2

unused2 ([ULONG](#))
Not used.

IMPL_EVCB Field - unused3

unused3 ([ULONG](#))
Not used.

JPEGOPTIONS

This structure contains fields used for setting JPEG-specific options through the JPEG I/O procedure. This structure is passed to the I/O procedure by setting the the *pExtraInfoStruct* field of [MMIOINFO](#) to the address of JPEGOPTIONS before passing the [MMIOINFO](#) structure to [mmioOpen](#). All unused fields must be set to zero.

```
typedef struct _JPEGOPTIONS {
    ULONG      ulStructLen;      /* Size of this structure. */
    USHORT     usQuantization[4]; /* Quantization value. */
    USHORT     usScale;          /* Scaling value. */
    ULONG      ulColorOrder;     /* Color order. */
    USHORT     usColorSpaceOut;  /* Subsampling value. */
} JPEGOPTIONS;

typedef JPEGOPTIONS *PJPEGOPTIONS;
```

JPEGOPTIONS Field - ulStructLen

ulStructLen (ULONG)
Size of this structure.

JPEGOPTIONS Field - usQuantization[4]

usQuantization[4] (USHORT)
An array of quantization values used for compression only. Each value has a range of 1-65535, where a value of 1 is the highest quality and a value of 65535 is the lowest quality. The default value is 100.

JPEGOPTIONS Field - usScale

usScale (USHORT)
A scaling value (used for decompression only) ranging from 1-8. A value of 1 produces image scaling of 1/8 the original size and a value of 8 produces no scaling (this is the default).

JPEGOPTIONS Field - ulColorOrder

ulColorOrder (ULONG)
A color order value (used for both compression and decompression) which can be set to one of the following values:

YUV_YVU	YCbCr color order (default)
~YUV_YVU	YCrCb color order

JPEGOPTIONS Field - usColorSpaceOut

usColorSpaceOut (USHORT)

A subsampling value (used for compression only) which can be set to one of the following values:

DST_YY (default)	Horizontal subsampling (uses BINT2X1.HDR)
DST_Y	No subsampling (uses BINT1X1.HDR)

LOCKH

This structure contains a lock handle for a memory object. It is used by stream handlers in the [SMH_LOCKMEM](#) message to lock or unlock a memory object.

```
typedef struct _LOCKH {
    BYTE    lock[16]; /* Lock handle. */
} LOCKH;

typedef LOCKH FAR *PLOCKH;
```

LOCKH Field - lock[16]

lock[16] (BYTE)

Contains the lock handle for a memory object.

LONG

Signed integer in the range -2 147 483 648 through 2 147 483 647.

```
#define LONG long
```

Note: Where this data type represents a graphic coordinate in world or model space, its value is restricted to -134 217 728 through 134 217 727.

A graphic coordinate in device or screen coordinates is restricted to -32 768 through 32 767.

The value of a graphic coordinate may be further restricted by any transforms currently in force, including the positioning of the origin of the window on the screen. In particular, coordinates in world or model space must not generate coordinate values after transformation (that is, in device or screen space) outside the range -32 768 through 32 767.

MASTER

This structure is used in the [SpiDetermineSyncMaster](#) call to identify the master streams to be used as the master stream of a synchronized

group.

```
typedef struct _MASTER {
    HSTREAM      hstreamMaster; /* Master stream handler. */
} MASTER;

typedef MASTER FAR *PMMASTER;
```

MASTER Field - hstreamMaster

hstreamMaster ([HSTREAM](#))

Contains the stream handler for the master stream.

MCI_AMP_GETDEVCAPS_PARMS

This structure contains parameters for the [MCI_GETDEVCAPS](#) message.

```
typedef struct _MCI_AMP_GETDEVCAPS_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG     ulReturn;      /* Return field. */
    ULONG     ulItem;        /* Not used. */
    USHORT    usMessage;     /* Message to query. */
    USHORT    usReserved0;   /* Reserved field. */
    ULONG     ulLength;      /* Length of structure. */
    ULONG     ulValue;       /* Value to determine capabilities. */
    ULONG     ulAttribute;   /* Flags modifying extended parms. */
    ULONG     ulExtended;    /* Flags. */
} MCI_AMP_GETDEVCAPS_PARMS;

typedef MCI_AMP_GETDEVCAPS_PARMS *PMCI_AMP_GETDEVCAPS_PARMS;
```

MCI_AMP_GETDEVCAPS_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_AMP_GETDEVCAPS_PARMS Field - ulReturn

ulReturn ([ULONG](#))

Return field.

MCI_AMP_GETDEVCAPS_PARMS Field - ullItem

ullItem (ULONG)
Not used.

MCI_AMP_GETDEVCAPS_PARMS Field - usMessage

usMessage (USHORT)
Field holds Media Control Interface message to query.

MCI_AMP_GETDEVCAPS_PARMS Field - usReserved0

usReserved0 (USHORT)
Reserved field.

MCI_AMP_GETDEVCAPS_PARMS Field - ulLength

ulLength (ULONG)
Length of structure.

MCI_AMP_GETDEVCAPS_PARMS Field - ulValue

ulValue (ULONG)
Value to determine capabilities. If *ulExtended* contains MCI_MIXER_LINE, *ulValue* should contain a valid media control interface connector. If *ulValue* is MCI_AMP_STREAM_CONNECTOR, the mixer will return if it can set the attribute globally.

MCI_AMP_GETDEVCAPS_PARMS Field - ulAttribute

ulAttribute (ULONG)
See the Amplifier Mixer Extensions of [MCI_GETDEVCAPS](#) for a list of flags that can be used to modify extended parameters.

MCI_AMP_GETDEVCAPS_PARMS Field - ulExtended

ulExtended ([ULONG](#))

Extended flags field. The following *ulExtended* value is defined:

MCI_MIXER_LINE

MCI_AMP_OPEN_PARMS

This structure contains fields for the [MCI_OPEN](#) message used with amplifier-mixer devices.

```
typedef struct _MCI_AMP_OPEN_PARMS {
    HWND      hwndCallback;    /* Window handle. */
    USHORT    usDeviceID;      /* Device ID. */
    USHORT    usReserved0;     /* Reserved field. */
    PSZ       pszDeviceType;    /* Device name from SYSTEM.INI. */
    PSZ       pszElementName;   /* File name or NULL. */
    PSZ       pszAlias;         /* Optional device alias. */
    PVOID     pDevDataPtr;     /* Device data. */
} MCI_AMP_OPEN_PARMS;
```

```
typedef MCI_AMP_OPEN_PARMS *PMCI_AMP_OPEN_PARMS;
```

MCI_AMP_OPEN_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle used for returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_AMP_OPEN_PARMS Field - usDeviceID

usDeviceID ([USHORT](#))

The device ID returned to the user.

MCI_AMP_OPEN_PARMS Field - usReserved0

usReserved0 ([USHORT](#))

Reserved field.

MCI_AMP_OPEN_PARMS Field - pszDeviceType

pszDeviceType ([PSZ](#))
The type of device.

MCI_AMP_OPEN_PARMS Field - pszElementName

pszElementName ([PSZ](#))
The media element or NULL.

MCI_AMP_OPEN_PARMS Field - pszAlias

pszAlias ([PSZ](#))
An optional device alias.

MCI_AMP_OPEN_PARMS Field - pDevDataPtr

pDevDataPtr ([PVOID](#))
A pointer to the amplifier-mixer device data structure or NULL.

MCI_AMP_SET_PARMS

This structure contains fields for the [MCI_SET](#) message for amp-mixer devices.

```
typedef struct _MCI_AMP_SET_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    ULONG      ulTimeFormat; /* Time format. */  
    ULONG      ulSpeedFormat; /* Speed format. */  
    ULONG      ulAudio; /* Audio attribute flags. */  
    ULONG      ulLevel; /* Level setting. */  
    ULONG      ulOver; /* Delay time. */  
    ULONG      ulItem; /* Item field. */  
    ULONG      ulValue; /* Connector. */  
} MCI_AMP_SET_PARMS;  
  
typedef MCI_AMP_SET_PARMS *PMCI_AMP_SET_PARMS;
```

MCI_AMP_SET_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_AMP_SET_PARMS Field - ulTimeFormat

ulTimeFormat ([ULONG](#))
The time format to be used by the device.

MCI_AMP_SET_PARMS Field - ulSpeedFormat

ulSpeedFormat ([ULONG](#))
The speed format to be used by the device.

MCI_AMP_SET_PARMS Field - ulAudio

ulAudio ([ULONG](#))
Audio attribute flags.

MCI_AMP_SET_PARMS Field - ulLevel

ulLevel ([ULONG](#))
Setting level as a percentage (0 - 100). A value of 0 represents full cutoff and a value of 100 represents full intensity.

MCI_AMP_SET_PARMS Field - ulOver

ulOver ([ULONG](#))

Delay time for vectored change, in milliseconds.

MCI_AMP_SET_PARMS Field - ullItem

ullItem ([ULONG](#))

Set to MCI_AMP_SET_AUDIO.

MCI_AMP_SET_PARMS Field - ulValue

ulValue ([ULONG](#))

This field contains a valid connector value. If *ulValue* contains MCI_AMP_STREAM_CONNECTOR, all modifications will affect the global output of the device.

MCI_BUFFER_PARMS

This structure contains fields for the [MCI_BUFFER](#) message.

```
typedef struct _MCI_BUFFER_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG      ulStructLength; /* Structure length. */
    ULONG      ulNumBuffers; /* Number of buffers. */
    ULONG      ulBufferSize; /* Suggested buffer size. */
    ULONG      ulMinToStart; /* Not used. */
    ULONG      ulSrcStart; /* Not used. */
    ULONG      ulTgtStart; /* Not used. */
    PVOID      pBufList; /* Pointer to list of buffers. */
} MCI_BUFFER_PARMS;

typedef MCI_BUFFER_PARMS *PMCI_BUFFER_PARMS;
```

MCI_BUFFER_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_BUFFER_PARMS Field - ulStructLength

ulStructLength (ULONG)
Structure length of MCI_BUFFER_PARMS.

MCI_BUFFER_PARMS Field - ulNumBuffers

ulNumBuffers (ULONG)
Number of buffers to allocate.

MCI_BUFFER_PARMS Field - ulBufferSize

ulBufferSize (ULONG)
Suggested size of buffers for MCI driver to use.

MCI_BUFFER_PARMS Field - ulMinToStart

ulMinToStart (ULONG)
Not used.

MCI_BUFFER_PARMS Field - ulSrcStart

ulSrcStart (ULONG)
Not used.

MCI_BUFFER_PARMS Field - ulTgtStart

ulTgtStart (ULONG)
Not used.

MCI_BUFFER_PARMS Field - pBufList

pBufList ([PVOID](#))

Pointer to list of buffers where the allocated information will be returned.

MCI_CAPTURE_PARMS

This structure contains fields for the [MCI_CAPTURE](#) message.

```
typedef struct _MCI_CAPTURE_PARMS {
    HWND     hwndCallback; /* Window handle. */
    RECTL     rect;        /* Rectangle area to capture. */
} MCI_CAPTURE_PARMS;

typedef MCI_CAPTURE_PARMS *PMCI_CAPTURE_PARMS;
```

MCI_CAPTURE_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_CAPTURE_PARMS Field - rect

rect ([RECTL](#))

A rectangle specifying the window subregion to be captured.

MCI_CDXA_SET_PARMS

This structure contains files for the [MCI_SET](#) message for CD/XA devices.

```
typedef struct _MCI_CDXA_SET_PARMS {
    HWND     hwndCallback; /* Window handle. */
    ULONG     ulTimeFormat; /* Device time format. */
    ULONG     ulSpeedFormat; /* Device speed format. */
    ULONG     ulAudio; /* Channel number for operation. */
    ULONG     ulLevel; /* Max level as percent. */
    ULONG     ulOver; /* Delay time. */
    ULONG     ulItem; /* Item field. */
    ULONG     ulValue; /* Item flag value. */
    ULONG     ulChannel; /* Channel number. */
    PVOID     pPlaylist; /* Pointer to playlist. */
    ULONG     ulPlaylistSize; /* Memory playlist size. */
} MCI_CDXA_SET_PARMS;
```

```
typedef MCI_CDXA_SET_PARMS *PMCI_CDXA_SET_PARMS;
```

MCI_CDXA_SET_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_CDXA_SET_PARMS Field - ulTimeFormat

ulTimeFormat ([ULONG](#))

The time format to be used by the device.

MCI_CDXA_SET_PARMS Field - ulSpeedFormat

ulSpeedFormat ([ULONG](#))

The speed format to be used by the device.

MCI_CDXA_SET_PARMS Field - ulAudio

ulAudio ([ULONG](#))

The channel number for the following operations:

- MCI_SET_AUDIO_LEFT
- MCI_SET_AUDIO_RIGHT
- MCI_SET_AUDIO_ALL

MCI_CDXA_SET_PARMS Field - ulLevel

ulLevel ([ULONG](#))

Audio volume level as a percentage of maximum.

MCI_CDXA_SET_PARMS Field - ulOver

ulOver (ULONG)
Delay time for vectored change, in milliseconds.

MCI_CDXA_SET_PARMS Field - ullItem

ullItem (ULONG)
Item field for set item flags.

MCI_CDXA_SET_PARMS Field - ulValue

ulValue (ULONG)
Value associated with item flag.

MCI_CDXA_SET_PARMS Field - ulChannel

ulChannel (ULONG)
Specifies which numeric CD/XA channel to set.

MCI_CDXA_SET_PARMS Field - pPlayList

pPlayList (PVOID)
Specifies a pointer to the memory playlist.

MCI_CDXA_SET_PARMS Field - ulPlayListSize

ulPlayListSize (ULONG)
Size of the memory playlist.

MCI_CDXA_STATUS_PARMS

This structure contains fields for the [MCI_STATUS](#) message for CD/XA devices.

```
typedef struct _MCI_CDXA_STATUS_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    ULONG     ulReturn;      /* Return field. */  
    ULONG     ulItem;        /* Status item field. */  
    ULONG     ulValue;       /* Status value field. */  
    ULONG     ulChannel;     /* Channel. */  
} MCI_CDXA_STATUS_PARMS;  
  
typedef MCI_CDXA_STATUS_PARMS *PMCI_CDXA_STATUS_PARMS;
```

MCI_CDXA_STATUS_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_CDXA_STATUS_PARMS Field - ulReturn

ulReturn ([ULONG](#))

Contains the status information upon return.

MCI_CDXA_STATUS_PARMS Field - ulItem

ulItem ([ULONG](#))

The status item to query.

MCI_CDXA_STATUS_PARMS Field - ulValue

ulValue ([ULONG](#))

Status value field.

MCI_CDXA_STATUS_PARMS Field - ulChannel

ulChannel ([ULONG](#))

Indicates the channel to test if MCI_CDXA_STATUS_CHANNEL is specified.

MCI_CONNECTION_PARMS

This structure contains fields for the [MCI_CONNECTION](#) message.

```
typedef struct _MCI_CONNECTION_PARMS {
    HWND      hwndCallback;    /* Window handle. */
    ULONG      ulConnectorType; /* Connector type. */
    ULONG      ulConnectorIndex; /* Connector index. */
    PSZ        pszAlias;        /* Secondary device alias. */
    USHORT     usToDeviceID;     /* ID of secondary device. */
    USHORT     usReserved0;      /* Reserved. */
} MCI_CONNECTION_PARMS;

typedef MCI_CONNECTION_PARMS *PMCI_CONNECTION_PARMS;
```

MCI_CONNECTION_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_CONNECTION_PARMS Field - ulConnectorType

ulConnectorType ([ULONG](#))

If specified, indicates that *ulConnectorIndex* is relative to the specified connector type. Otherwise, *ulConnectorIndex* is an absolute index.

MCI_CONNECTION_PARMS Field - ulConnectorIndex

ulConnectorIndex ([ULONG](#))

Connector index for the primary device. The interpretation of the connector index depends on the *ulConnectorType* field. If the MCI_CONNECTOR_INDEX is not specified then the first connector is assumed.

MCI_CONNECTION_PARMS Field - pszAlias

pszAlias (PSZ)

The alias name to set for the secondary device. If the alias already exists for another device the error MCIERR_DUPLICATE_ALIAS is returned. If the connected to device already has an alias the error MCIERR_CANNOT_ALIAS is returned.

MCI_CONNECTION_PARMS Field - usToDeviceID

usToDeviceID (USHORT)

The device ID of the secondary device to which the connection is being established.

MCI_CONNECTION_PARMS Field - usReserved0

usReserved0 (USHORT)

Reserved for future use. Must be zero.

MCI_CONNECTOR_PARMS

This structure contains fields for the [MCI_CONNECTOR](#) message.

```
typedef struct _MCI_CONNECTOR_PARMS {  
    HWND     hwndCallback;    /* Window handle. */  
    ULONG     ulReturn;        /* Return information. */  
    ULONG     ulConnectorType; /* Connector type. */  
    ULONG     ulConnectorIndex; /* Connector number. */  
} MCI_CONNECTOR_PARMS;
```

```
typedef MCI_CONNECTOR_PARMS *PMCI_CONNECTOR_PARMS;
```

MCI_CONNECTOR_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_CONNECTOR_PARMS Field - ulReturn

ulReturn (ULONG)

Return information.

MCI_CONNECTOR_PARDS Field - ulConnectorType

ulConnectorType (ULONG)

If specified, indicates that *ulConnectorIndex* is relative to the specified connector type. Otherwise, *ulConnectorIndex* is an absolute index.

MCI_CONNECTOR_PARDS Field - ulConnectorIndex

ulConnectorIndex (ULONG)

The connector number to set or query. The interpretation of the connector number is dependent on the *ulConnectorType* field.

MCI_CONNECTORINFO_PARDS

This structure contains fields for the [MCI_CONNECTORINFO](#) message.

```
typedef struct _MCI_CONNECTORINFO_PARDS {
    HWND     hwndCallback;      /* Window handle. */
    ULONG     ulReturn;         /* Return information. */
    ULONG     ulDeviceTypeID;   /* Device type. */
    ULONG     ulConnectorType;  /* Connector type. */
    ULONG     ulConnectorIndex; /* Connector index. */
    ULONG     ulToConnectorType; /* Connector type to test. */
} MCI_CONNECTORINFO_PARDS;

typedef MCI_CONNECTORINFO_PARDS *PMCI_CONNECTORINFO_PARDS;
```

MCI_CONNECTORINFO_PARDS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_CONNECTORINFO_PARDS Field - ulReturn

ulReturn (ULONG)

Return information.

MCI_CONNECTORINFO_PARMS Field - ulDeviceTypeID

ulDeviceTypeID (ULONG)

Indicates the media control interface device type for which connector information is to be returned.

MCI_CONNECTORINFO_PARMS Field - ulConnectorType

ulConnectorType (ULONG)

If specified, indicates that *ulConnectorIndex* is relative to the specified connector type. Otherwise, *ulConnectorIndex* is an absolute index.

MCI_CONNECTORINFO_PARMS Field - ulConnectorIndex

ulConnectorIndex (ULONG)

Connector index for the primary device. The interpretation of the connector index depends on the *ulConnectorType* field. If the MCI_CONNECTOR_INDEX flag is not specified then the first connector is assume.

MCI_CONNECTORINFO_PARMS Field - ulToConnectorType

ulToConnectorType (ULONG)

Indicates the connector type to test if MCI_QUERY_VALID_CONNECTION is specified. Returns MCI_TRUE if the connector type specified in *ulConnectorType* and *ulToConnectorType* are compatible, otherwise returns MCI_FALSE.

MCI_CUEPOINT_PARMS

This structure contains fields for setting run-time cue points through the [MCI_SET_CUEPOINT](#) message.

```
typedef struct _MCI_CUEPOINT_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG     ulCuepoint;    /* Cue point location. */
    USHORT    usUserParm;    /* User parameter returned. */
    USHORT    usReserved0;   /* Reserved. */
} MCI_CUEPOINT_PARMS;
```

```
typedef MCI_CUEPOINT_PARMS *PMCI_CUEPOINT_PARMS;
```

MCI_CUEPOINT_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_CUEPOINT_PARMS Field - ulCuepoint

ulCuepoint ([ULONG](#))

The cue point location in the current time format.

MCI_CUEPOINT_PARMS Field - usUserParm

usUserParm ([USHORT](#))

The user parameter returned for the cue point asynchronous notification message ([MM_MCICUEPOINT](#)).

MCI_CUEPOINT_PARMS Field - usReserved0

usReserved0 ([USHORT](#))

Reserved.

MCI_DEFAULT_CONNECTION_PARMS

This structure contains fields for the [MCI_DEFAULT_CONNECTION](#) message.

```
typedef struct _MCI_DEFAULT_CONNECTION_PARMS {
    HWND      hwndCallback;      /* Window handle. */
    PSZ       pszDevice;         /* Device name. */
    ULONG     ulConnectorType;    /* Connector type. */
    ULONG     ulConnectorIndex;   /* Connector index. */
    PSZ       pszToDevice;        /* Returns device name. */
    ULONG     ulToConnectorType;  /* Connector type. */
    ULONG     ulToConnectorIndex; /* Connector number. */
} MCI_DEFAULT_CONNECTION_PARMS;

typedef MCI_DEFAULT_CONNECTION_PARMS *PMCI_DEFAULT_CONNECTION_PARMS;
```

MCI_DEFAULT_CONNECTION_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_DEFAULT_CONNECTION_PARMS Field - pszDevice

pszDevice (PSZ)

Device name.

MCI_DEFAULT_CONNECTION_PARMS Field - ulConnectorType

ulConnectorType (ULONG)

If specified, indicates that *ulConnectorIndex* is relative to the specified connector type. Otherwise, *ulConnectorIndex* is an absolute index.

MCI_DEFAULT_CONNECTION_PARMS Field - ulConnectorIndex

ulConnectorIndex (ULONG)

Connector index for the primary device. The interpretation of the connector index depends on the *ulConnectorType* field. If MCI_CONNECTOR_INDEX is not specified, then the first connector is assumed.

MCI_DEFAULT_CONNECTION_PARMS Field - pszToDevice

pszToDevice (PSZ)

The user-application must allocate memory for the device name. If MCI_MAKE_CONNECTION is specified, this field indicates the device name to which the connection is to be established. If MCI_QUERY_CONNECTION is specified, and a connection exists, this field returns the device name to which the connection exists. If no connection exists, this field is set to NULL on return and the error MCIERR_NO_CONNECTION is returned.

MCI_DEFAULT_CONNECTION_PARMS Field - ulToConnectorType

ulToConnectorType (ULONG)

If MCI_CONNECTOR_TYPE is specified, this field indicates the connector type of the connector on the other device for this connection. This field indicates the connector type on input if MCI_MAKE_CONNECTION is specified, and indicates the connector type on output if MCI_QUERY_CONNECTION is specified.

MCI_DEFAULT_CONNECTION_PARMS Field - ulToConnectorIndex

ulToConnectorIndex (ULONG)

Specifies the connector number of the connector on the other device for this connection. This value is specified on input when establishing a connection, and is returned on output when querying a connection. This connector index is either absolute or relative, depending on whether or not the MCI_CONNECTOR_TYPE flag is set. If MCI_CONNECTOR_INDEX is not specified, 1 is assumed.

MCI_DEVICESETTINGS_PARMS

This structure contains fields for the [MCI_DEVICESETTINGS](#) message.

```
typedef struct _MCI_DEVICESETTINGS_PARMS {
    HWND      hwndCallback; /* Window handle. */
    HWND      hwndNotebook; /* Handle to notebook window. */
    USHORT    usDeviceType; /* Device type. */
    PSZ       pszDeviceName; /* Logical device name. */
} MCI_DEVICESETTINGS_PARMS;

typedef MCI_DEVICESETTINGS_PARMS *PMCI_DEVICESETTINGS_PARMS;
```

MCI_DEVICESETTINGS_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_DEVICESETTINGS_PARMS Field - hwndNotebook

hwndNotebook ([HWND](#))
Handle to notebook window.

MCI_DEVICESETTINGS_PARMS Field - usDeviceType

usDeviceType ([USHORT](#))
Device type.

MCI_DEVICESETTINGS_PARMS Field - pszDeviceName

pszDeviceName ([PSZ](#))
Logical device name.

MCI_DGV_OPEN_PARMS

This structure is the same as [MCI_VID_OPEN_PARMS](#).

```
typedef MCI_VID_OPEN_PARMS MCI_DGV_OPEN_PARMS;
```

MCI_DGV_PLAY_PARMS

This structure contains fields for the [MCI_PLAY](#) message for digital video devices.

```
typedef struct _MCI_DGV_PLAY_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    ULONG      ulFrom;       /* Starting play position. */  
    ULONG      ulTo;         /* Ending play position. */  
    ULONG      ulSpeed;      /* Frames per second play speed. */  
} MCI_DGV_PLAY_PARMS;  
  
typedef MCI_DGV_PLAY_PARMS *PMCI_DGV_PLAY_PARMS;
```

MCI_DGV_PLAY_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_DGV_PLAY_PARMS Field - ulFrom

ulFrom ([ULONG](#))
The position to play from.

MCI_DGV_PLAY_PARMS Field - ulTo

ulTo ([ULONG](#))
The position to play to.

MCI_DGV_PLAY_PARMS Field - ulSpeed

ulSpeed ([ULONG](#))
The play rate in frames per second.

MCI_DGV_RECT_PARMS

This structure is the same as [MCI_VID_RECT_PARMS](#).

```
typedef MCI_VID_RECT_PARMS MCI_DGV_RECT_PARMS;
```

MCI_DGV_SET_PARMS

This structure contains fields for the [MCI_SET](#) message for digital video devices. This structure is the same as [MCI_SET_PARMS](#).

```
typedef MCI_SET_PARMS MCI_DGV_SET_PARMS;
```

MCI_DGV_TUNER_PARMS

This structure contains fields for the [MCI_SETTUNER](#) message for digital video devices.


```
typedef struct _MCI_DGV_TUNER_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG     ulFrequency; /* Tuner frequency. */
    ULONG     ulReserved0; /* Reserved. */
    ULONG     ulTVChannel; /* TV channel. */
    LONG      lFineTune; /* Fine tuning adjustments. */
    PSZ       pszRegion; /* TV channel region. */
    ULONG     ulReserved1; /* Reserved. */
    ULONG     ulReserved2; /* Reserved. */
} MCI_DGV_TUNER_PARMS;

typedef MCI_DGV_TUNER_PARMS *PMCI_DGV_TUNER_PARMS;
```

MCI_DGV_TUNER_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle to be used in returning asynchronous notification. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_DGV_TUNER_PARMS Field - ulFrequency

ulFrequency ([ULONG](#))
Tuner frequency.

MCI_DGV_TUNER_PARMS Field - ulReserved0

ulReserved0 ([ULONG](#))
This field is reserved for future use.

MCI_DGV_TUNER_PARMS Field - ulTVChannel

ulTVChannel ([ULONG](#))
TV channel.

MCI_DGV_TUNER_PARMS Field - lFineTune

IFineTune ([LONG](#))
Fine tuning adjustments.

MCI_DGV_TUNER_PARMS Field - pszRegion

pszRegion ([PSZ](#))
TV channel region.

MCI_DGV_TUNER_PARMS Field - ulReserved1

ulReserved1 ([ULONG](#))
This field is reserved for future use.

MCI_DGV_TUNER_PARMS Field - ulReserved2

ulReserved2 ([ULONG](#))
This field is reserved for future use.

MCI_DGV_WINDOW_PARMS

This structure contains fields for the [MCI_WINDOW](#) message for digital video devices. This structure is the same as [MCI_VID_WINDOW_PARMS](#).

```
typedef MCI_VID_WINDOW_PARMS MCI_DGV_WINDOW_PARMS;
```

MCIDRV_CHANGERESOURCE_PARMS

This structure contains parameters for the [MCIDRV_CHANGERESOURCE](#) message. This message is sent from MCDs to MDM. This is not an application data structure.

```
typedef struct _MCIDRV_CHANGERESOURCE_PARMS {  
    PVOID      pInstance;          /* Pointer to device instance. */  
    USHORT     usResourceUnits;    /* Required resource units. */  
    USHORT     usResourceClass;    /* Resource class. */  
    USHORT     usResourcePriority; /* Reserved for future use. */  
} MCIDRV_CHANGERESOURCE_PARMS;  
  
typedef MCIDRV_CHANGERESOURCE_PARMS *PMCIDRV_CHANGERESOURCE_PARMS;
```

MCIDRV_CHANGERESOURCE_PARMS Field - pInstance

pInstance ([PVOID](#))
Pointer to device context structure.

MCIDRV_CHANGERESOURCE_PARMS Field - usResourceUnits

usResourceUnits ([USHORT](#))
New resource units.

MCIDRV_CHANGERESOURCE_PARMS Field - usResourceClass

usResourceClass ([USHORT](#))
New resource class.

MCIDRV_CHANGERESOURCE_PARMS Field - usResourcePriority

usResourcePriority ([USHORT](#))
Reserved for future use.

MCI_EDIT_PARMS

This structure contains fields for the [MCI_CUT](#), [MCI_PASTE](#), and [MCI_COPY](#) messages.

```
typedef struct _MCI_EDIT_PARMS {  
    HWND     hwndCallback; /* Window handle. */  
    ULONG     ulStructLen;  /* Length of the structure. */  
    ULONG     ulFrom;       /* Beginning point of range. */  
    ULONG     ulTo;         /* Ending point of range. */  
};
```

```
PVOID    pBuff;           /* Application buffer. */
ULONG    ulBufLen;        /* Length of application buffer. */
PVOID    pHeader;         /* Header describing buffer. */
} MCI_EDIT_PARMS;

typedef MCI_EDIT_PARMS *PMCI_EDIT_PARMS;
```

Note: For MCI_CUT, MCI_COPY, MCI_PASTE; if MCI_BUFFER is passed in, then *pBuff* must contain a valid pointer.

MCI_EDIT_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_EDIT_PARMS Field - ulStructLen

ulStructLen ([ULONG](#))

Length of the structure.

MCI_EDIT_PARMS Field - ulFrom

ulFrom ([ULONG](#))

Beginning point of range in device element.

MCI_EDIT_PARMS Field - ulTo

ulTo ([ULONG](#))

Ending point of range in device element.

MCI_EDIT_PARMS Field - pBuff

pBuff ([PVOID](#))

Pointer to an application-supplied buffer; otherwise, NULL to indicate clipboard.

MCI_EDIT_PARMS Field - ulBufLen

ulBufLen ([ULONG](#))
Length of application buffer.

MCI_EDIT_PARMS Field - pHeader

pHeader ([PVOID](#))
Header that describes the application-supplied buffer.

MCI_ESCAPE_PARMS

This structure contains fields for the [MCI_ESCAPE](#) message.

```
typedef struct _MCI_ESCAPE_PARMS {  
    HWND    hwndCallback; /* Window handle. */  
    PSZ     pszCommand;   /* Pointer to null terminated buffer. */  
} MCI_ESCAPE_PARMS;  
  
typedef MCI_ESCAPE_PARMS *PMCI_ESCAPE_PARMS;
```

MCI_ESCAPE_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_ESCAPE_PARMS Field - pszCommand

pszCommand ([PSZ](#))
A pointer to a null-terminated buffer that contains the command to send to the device.

MCI_GENERIC_PARMS

This structure contains fields for messages that have empty parameter lists.

```
typedef struct _MCI_GENERIC_PARMS {  
    HWND     hwndCallback; /* Window handle. */  
} MCI_GENERIC_PARMS;  
  
typedef MCI_GENERIC_PARMS *PMCI_GENERIC_PARMS;
```

MCI_GENERIC_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_GETDEVCAPS_PARMS

This structure contains fields for the [MCI_GETDEVCAPS](#) message.

```
typedef struct _MCI_GETDEVCAPS_PARMS {  
    HWND     hwndCallback; /* Window handle. */  
    ULONG     ulReturn;     /* Return information. */  
    ULONG     ulItem;       /* Additional capability. */  
    USHORT    usMessage;    /* Message ID. */  
    USHORT    usReserved0;  /* Reserved. */  
} MCI_GETDEVCAPS_PARMS;  
  
typedef MCI_GETDEVCAPS_PARMS *PMCI_GETDEVCAPS_PARMS;
```

MCI_GETDEVCAPS_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_GETDEVCAPS_PARMS Field - ulReturn

ulReturn (ULONG)

The format of the *ulReturn* value in this structure is defined by the high-order word of the value returned by [mciSendCommand](#). This value is used by [mciSendString](#) to determine how to convert the *ulReturn* value to string form. See the MMDRVOS2.H header file for a list of possible format values.

MCI_GETDEVCAPS_PARMS Field - ullItem

ullItem ([ULONG](#))

A ULONG identifying the device capability to be queried if MCI_GETDEVCAPS_ITEM is specified.

MCI_GETDEVCAPS_PARMS Field - usMessage

usMessage ([USHORT](#))

The message ID being queried if MCI_GETDEVCAPS_MESSAGE is specified.

MCI_GETDEVCAPS_PARMS Field - usReserved0

usReserved0 ([USHORT](#))

Reserved.

MCI_GROUP_PARMS

This structure contains fields for the [MCI_GROUP](#) message.

```
typedef struct _MCI_GROUP_PARMS {
    HWND      hwndCallback;    /* Window handle. */
    USHORT    usGroupID;       /* Group ID returned. */
    USHORT    usReserved0;     /* Reserved. */
    ULONG     ulStructLength;   /* Length of structure. */
    USHORT    usMasterID;      /* ID of master device. */
    USHORT    usReserved1;     /* Reserved. */
    PSZ       pszGroupAlias;    /* Alias name. */
    ULONG     ulNumDevices;     /* Number of devices in group. */
    PULONG    pDevDeviceID;     /* Device IDs in the group. */
} MCI_GROUP_PARMS;

typedef MCI_GROUP_PARMS *PMCI_GROUP_PARMS;
```

MCI_GROUP_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_GROUP_PARMS Field - usGroupID

usGroupID (USHORT)
This field contains the unique identification number used to distinguish one particular group from other groups/devices/instances. On an MCI_GROUP_MAKE, this field will be filled in and returned to the caller. On MCI_GROUP_DELETE, this field will be used to reference the specific group to delete if no MCI_GROUP_ALIAS flag is specified.

MCI_GROUP_PARMS Field - usReserved0

usReserved0 (USHORT)
Reserved.

MCI_GROUP_PARMS Field - ulStructLength

ulStructLength (ULONG)
This is the length of the data structure from the first field (*hwndCallback*) to the last field (*paulDeviceID*). If the structure changes in the future, MDM can process the data according to this value.

MCI_GROUP_PARMS Field - usMasterID

usMasterID (USHORT)
This field is currently unused. It will be used in the future for group synchronization.

MCI_GROUP_PARMS Field - usReserved1

usReserved1 (USHORT)
Reserved.

MCI_GROUP_PARMS Field - pszGroupAlias

pszGroupAlias ([PSZ](#))

A pointer to an ASCIIZ string containing the alias name for this particular group. This field is processed only when the MCI_GROUP_ALIAS and the MCI_GROUP_MAKE flags are set.

MCI_GROUP_PARMS Field - ulNumDevices

ulNumDevices ([ULONG](#))

Number of devices in group.

MCI_GROUP_PARMS Field - paulDeviceID

paulDeviceID ([PULONG](#))

This field contains an array of device IDs in the group.

MCI_IMAGE_PARMS

This structure contains fields for the [MCI_GETIMAGEBUFFER](#) and [MCI_SETIMAGEBUFFER](#) messages.

```
typedef struct _MCI_IMAGE_PARMS {
    HWND      hwndCallback;      /* Window handle. */
    ULONG      ulPelFormat;       /* Pel format. */
    USHORT     usBitCount;        /* Number of bits. */
    USHORT     usReserved0;       /* Reserved. */
    ULONG      ulImageCompression; /* Image compression. */
    RECT       rect;              /* Rectangle. */
    PVOID      pPelBuffer;        /* Pointer to buffer. */
    ULONG      ulBufferHeight;    /* Buffer height. */
    ULONG      ulBufferWidth;     /* Buffer width. */
    ULONG      ulBufLen;          /* Buffer length. */
} MCI_IMAGE_PARMS;

typedef MCI_IMAGE_PARMS *PMCI_IMAGE_PARMS;
```

MCI_IMAGE_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_IMAGE_PARMS Field - ulPelFormat

ulPelFormat (ULONG)

The format of the data in the capture device element. If MCI_CONVERT is TRUE, this field contains one of the supported OS/2 2.0 bitmap formats. If MCI_CONVERT is FALSE, this field contains a device-specific format. The specified format must be compatible with the value in the *usBitCount* field.

For MCI_GETIMAGEBUFFER, the valid pel format values for hardware include the default format of the card or an OS/2 bitmap. For a movie, the only valid value is RGB or an OS/2 bitmap.

This field is input for MCI_SETIMAGEBUFFER, and output for MCI_GETIMAGEBUFFER.

If a palettized format is requested, the currently defined palette will be supplied by MCI_SETIMAGEPALETTE.

MCI_IMAGE_PARMS Field - usBitCount

usBitCount (USHORT)

Number of bits per pel for the data in the capture device element. If MCI_CONVERT is true, this field contains a value for one of the supported OS/2 bitmap pel formats (1, 4, 8, or 24). If MCI_CONVERT is FALSE, this field contains a device-specific value.

MCI_IMAGE_PARMS Field - usReserved0

usReserved0 (USHORT)

Reserved.

MCI_IMAGE_PARMS Field - ullImageCompression

ullImageCompression (ULONG)

Compression type of data in buffer.

M-Motion specific: This is MCI_IMG_COMP_NONE.

MCI_IMAGE_PARMS Field - rect

rect (RECTL)

A window rectangle of the area to read or write.

M-Motion specific: Not supported.

MCI_IMAGE_PARMS Field - pPelBuffer

pPelBuffer ([PVOID](#))

If *pPelBuffer* is equal to 0 on an [MCI_GETIMAGEBUFFER](#) message, the message will be considered a query for the buffer length and attributes of the current image element.

MCI_IMAGE_PARMS Field - ulBufferHeight

ulBufferHeight ([ULONG](#))

Height of the buffer specified by *pPelBuffer*.

MCI_IMAGE_PARMS Field - ulBufferWidth

ulBufferWidth ([ULONG](#))

Width of the buffer specified by *pPelBuffer*.

MCI_IMAGE_PARMS Field - ulBufLen

ulBufLen ([ULONG](#))

Length of buffer specified by *pPelBuffer*.

MCI_INFO_PARMS

This structure contains fields for the [MCI_INFO](#) message.

```
typedef struct _MCI_INFO_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    PSZ       pszReturn;    /* Return string. */  
    ULONG     ulRetSize;    /* Return string size. */  
} MCI_INFO_PARMS;
```

```
typedef MCI_INFO_PARMS *PMCI_INFO_PARMS;
```

MCI_INFO_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_INFO_PARMS Field - pszReturn

pszReturn ([PSZ](#))

A pointer to a user-supplied buffer for the return string.

MCI_INFO_PARMS Field - ulRetSize

ulRetSize ([ULONG](#))

The size, in bytes, of the buffer for the return string. Upon return it contains the number of bytes the string required. If the size was greater or equal to this size, the entire string will be returned; otherwise, only what will fit is returned.

MCI_LOAD_PARMS

This structure contains fields for the [MCI_LOAD](#) message.

```
typedef struct _MCI_LOAD_PARMS {  
    HWND    hwndCallback; /* Window handle. */  
    PSZ     pszElementName; /* Element name. */  
} MCI_LOAD_PARMS;  
  
typedef MCI_LOAD_PARMS *PMCI_LOAD_PARMS;
```

MCI_LOAD_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_LOAD_PARMS Field - pszElementName

pszElementName ([PSZ](#))

The media element to load.

MCI_MASTERAUDIO_PARMS

This structure contains fields for the [MCI_MASTERAUDIO](#) message.

```
typedef struct _MCI_MASTERAUDIO_PARMS {
    HWND    hwndDummyCallback; /* Notification is not supported. */
    ULONG    ulReturn;         /* Return. */
    ULONG    ulMasterVolume;   /* Master volume setting. */
} MCI_MASTERAUDIO_PARMS;

typedef MCI_MASTERAUDIO_PARMS *PMCI_MASTERAUDIO_PARMS;
```

MCI_MASTERAUDIO_PARMS Field - hwndDummyCallback

hwndDummyCallback ([HWND](#))

The [MCI_MASTERAUDIO](#) message does not support notification messages.

MCI_MASTERAUDIO_PARMS Field - ulReturn

ulReturn ([ULONG](#))

Master volume level or speaker/headphone on/off (MCI_TRUE/MCI_FALSE) state if MCI_QUERYCURRENTSETTING or MCI_QUEYSAVEDSETTING is specified.

MCI_MASTERAUDIO_PARMS Field - ulMasterVolume

ulMasterVolume ([ULONG](#))

The master volume level setting. If MCI_MASTERVOL is specified, this field contains the master volume as a percentage. If a value greater than 100 is given, 100 will be used, and no error will be returned.

MCI_MIX_BUFFER

This structure contains fields for the [MCI_BUFFER](#) message. It is used for both the mixWrite and mixRead entry points. In addition, when the mixer calls the application back, it will use an MCI_MIX_BUFFER pointer.

```
typedef struct _MCI_MIX_BUFFER {
    ULONG    ulStructLength; /* Structure length. */
    PVOID    pBuffer;        /* Pointer to a buffer. */
    ULONG    ulBufferLength; /* Length of the buffer. */
    ULONG    ulFlags;        /* Flags. */
}
```

```
ULONG      ulUserParm;        /* User parameter. */
ULONG      ulTime;           /* Device time in ms. */
ULONG      ulReserved1;      /* Reserved. */
ULONG      ulReserved2;      /* Reserved. */
} MCI_MIX_BUFFER;

typedef MCI_MIX_BUFFER *PMCI_MIX_BUFFER;
```

MCI_MIX_BUFFER Field - ulStructLength

ulStructLength (ULONG)
Structure length of MCI_MIX_BUFFER.

MCI_MIX_BUFFER Field - pBuffer

pBuffer (PVOID)
Pointer to a buffer.

MCI_MIX_BUFFER Field - ulBufferLength

ulBufferLength (ULONG)
Length of the buffer.

MCI_MIX_BUFFER Field - ulFlags

ulFlags (ULONG)
The only valid value for *ulFlags* is MIX_BUFFER_EOS. This field should be filled in by the application when the last buffer is sent to the mixer.

MCI_MIX_BUFFER Field - ulUserParm

ulUserParm (ULONG)
User parameter associated with the buffer. This field allows a specific value to be attached to each buffer. When an individual buffer is returned, the *ulUserParm* associated with the buffer is also returned.

MCI_MIX_BUFFER Field - ulTime

ulTime (ULONG)

Device time in milliseconds. The current time of the device is returned with every buffer and is placed in the *ulTime* field. Applications can use this for synchronization purposes. (Timing information can also be obtained using [MCI_STATUS](#) and the [MCI_STATUS_POSITION](#) flag.)

MCI_MIX_BUFFER Field - ulReserved1

ulReserved1 (ULONG)

Reserved for future use and must be set to zero.

MCI_MIX_BUFFER Field - ulReserved2

ulReserved2 (ULONG)

Reserved for future use and must be set to zero.

MCI_MIXEVENT_PARMS

This structure contains device-specific event data for [MM_MCIEVENT](#) messages generated by the [MCI_MIXNOTIFY](#) message.

```
typedef struct _MCI_MIXEVENT_PARMS {
    ULONG        ulLength;           /* Length of structure. */
    HWND         hwndMixer;          /* Window to inform of mixer changes. */
    ULONG        ulFlags;            /* Either connector or attribute change. */
    USHORT       usDeviceID;         /* Device ID. */
    ULONG        ulDeviceType;       /* Device type. */
    ULONG        ulDeviceOrdinal;    /* Ordinal of device type. */
    ULONG        ulAttribute;        /* Attribute that changed. */
    ULONG        ulValue;           /* New value of the changed attribute. */
    ULONG        ulConnectorType;    /* Connector type. */
    ULONG        ulConnectorIndex;   /* Connector index. */
    ULONG        ulConnStatus;       /* Connector status. */
} MCI_MIXEVENT_PARMS;

typedef MCI_MIXEVENT_PARMS *PMCI_MIXEVENT_PARMS;
```

MCI_MIXEVENT_PARMS Field - ulLength

ulLength (ULONG)
Length of structure.

MCI_MIXEVENT_PARMS Field - hwndMixer

hwndMixer (HWND)
Window to inform of mixer changes.

MCI_MIXEVENT_PARMS Field - ulFlags

ulFlags (ULONG)
Contains one of the following values:

MCI_MIX_ATTRIBUTE
Indicates that *ulAttribute*, *ulDeviceType*, and *ulValue* fields are valid.

MCI_MIX_CONNECTOR
Indicates that a connector has been changed and the *ulConnectorType*, *ulConnectorIndex*, and *ulConnStatus* fields are valid.

MCI_MIXEVENT_PARMS Field - usDeviceID

usDeviceID (USHORT)
The device ID to notify of the change.

MCI_MIXEVENT_PARMS Field - ulDeviceType

ulDeviceType (ULONG)
Device type that generated the change.

MCI_MIXEVENT_PARMS Field - ulDeviceOrdinal

ulDeviceOrdinal (ULONG)
Ordinal of device type.

MCI_MIXEVENT_PARMS Field - ulAttribute

ulAttribute ([ULONG](#))
Attribute that changed.

MCI_MIXEVENT_PARMS Field - ulValue

ulValue ([ULONG](#))
New value of the changed attribute.

MCI_MIXEVENT_PARMS Field - ulConnectorType

ulConnectorType ([ULONG](#))
Connector type.

MCI_MIXEVENT_PARMS Field - ulConnectorIndex

ulConnectorIndex ([ULONG](#))
Connector index.

MCI_MIXEVENT_PARMS Field - ulConnStatus

ulConnStatus ([ULONG](#))
Connector status; enabled or disabled.

MCI_MIXSETUP_PARMS

This structure contains fields for the [MCI_MIXSETUP](#) message.

```
typedef struct _MCI_MIXSETUP_PARMS {  
    HWND         hwndCallback;    /* Window handle. */  
};
```

```

ULONG        ulBitsPerSample; /* Bits per sample. */
ULONG        ulFormatTag;     /* Format tag. */
ULONG        ulSamplesPerSec; /* Sampling rate. */
ULONG        ulChannels;      /* Number of channels. */
ULONG        ulFormatMode;    /* Play or record. */
ULONG        ulDeviceType;    /* Device type. */
ULONG        ulMixHandle;     /* Mixer handle. */
PMIXERPROC   pmixWrite;       /* Entry point. */
PMIXERPROC   pmixRead;        /* Entry point. */
PMIXEREVENT  pmixEvent;       /* Entry point. */
PVOID        pExtendedInfo;   /* Extended information. */
ULONG        ulBufferSize;    /* Recommended buffer size. */
ULONG        ulNumBuffers;    /* Recommended number of buffers. */
} MCI_MIXSETUP_PARMS;

typedef MCI_MIXSETUP_PARMS *PMCI_MIXSETUP_PARMS;

```

MCI_MIXSETUP_PARMS Field - hwndCallback

hwndCallback (**HWND**) - input

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_MIXSETUP_PARMS Field - ulBitsPerSample

ulBitsPerSample (**ULONG**) - input

Number of bits per sample.

MCI_MIXSETUP_PARMS Field - ulFormatTag

ulFormatTag (**ULONG**) - input

The following are valid format values. These formats are not supported on all audio devices. Additional subtype values can be found in OS2MEDEF.H.

MCI_WAVE_FORMAT_PCM
Changes the format to pulse code modulation (PCM).

MCI_WAVE_FORMAT_ADPCM
Changes the format to adaptive differential pulse code modulation (ADPCM).

MCI_WAVE_FORMAT_IBM_CVSD
Changes the format to IBM Speech Viewer.

MCI_WAVE_FORMAT_ALAW
Changes the format to A-Law.

MCI_WAVE_FORMAT_MULAW
Changes the format to Mu-Law.

MCI_WAVE_FORMAT_IBM_ALAW
Changes the format to A-Law.

MCI_WAVE_FORMAT_IBM_MULAW
Changes the format to Mu-Law.

MCI_WAVE_FORMAT_OKI_ADPCM
Changes the format to OKI ADPCM.

MCI_WAVE_FORMAT_DVI_ADPCM
Changes the format to DVI ADPCM.

MCI_WAVE_FORMAT_IBM_ADPCM
Changes the format to ADPCM.

MCI_WAVE_FORMAT_DIGISTD
Changes the format to IBM Digispeech (standard format).

MCI_WAVE_FORMAT_DIGIFIX
Changes the format to IBM Digispeech (fixed format).

MCI_WAVE_FORMAT_AVC_ADPCM
Changes the format to AVC ADPCM.

MCI_WAVE_FORMAT_CT_ADPCM
Changes the format to Creative Labs ADPCM.

MCI_WAVE_FORMAT_MPEG1
Changes the format to MPEG audio.

MCI_MIXSETUP_PARMS Field - ulSamplesPerSec

ulSamplesPerSec (ULONG) - input
Sampling rate.

MCI_MIXSETUP_PARMS Field - ulChannels

ulChannels (ULONG) - input
Number of channels.

MCI_MIXSETUP_PARMS Field - ulFormatMode

ulFormatMode (ULONG) - input
Specifying MCI_PLAY or MCI_RECORD sets up the mixer device to play or record accordingly.

MCI_MIXSETUP_PARMS Field - ulDeviceType

ulDeviceType (ULONG) - input

Device type can be one of the following values:

- MCI_DEVTTYPE_WAVEFORM_AUDIO
- MCI_DEVTTYPE_SEQUENCER

If *ulDeviceType* is MCI_DEVTTYPE_WAVEFORM_AUDIO, the following digital audio fields must be specified: *ulBitsPerSample*, *ulFormatTag*, *ulSamplesPerSec*, and *ulChannels*.

MCI_MIXSETUP_PARMS Field - ulMixHandle

ulMixHandle (ULONG) - output

The handle returned by the mixer to enable communication with the application.

MCI_MIXSETUP_PARMS Field - pmixWrite

pmixWrite (PMIXERPROC) - output

Write routine entry point. Upon successful completion of the *pmixEvent* routine, the mixer updates this field so that the application can write buffers to the mixer.

MCI_MIXSETUP_PARMS Field - pmixRead

pmixRead (PMIXERPROC) - output

Read routine entry point. Upon successful completion of the *pmixEvent* routine, the mixer updates this field so that the application can read buffers from the mixer.

MCI_MIXSETUP_PARMS Field - pmixEvent

pmixEvent (PMIXEREVENT) - input

Mixer event routine entry point. When the mixer has finished reading or writing a buffer it will call this function.

MCI_MIXSETUP_PARMS Field - pExtendedInfo

pExtendedInfo (PVOID)

Pointer to extended information. Some media types (such as compressed audio) require additional information to be passed to the wave driver. This field is used to pass this information to the mixer.

MCI_MIXSETUP_PARMS Field - ulBufferSize

ulBufferSize ([ULONG](#)) - output
Recommended buffer size according to the requested setup.

MCI_MIXSETUP_PARMS Field - ulNumBuffers

ulNumBuffers ([ULONG](#)) - output
Recommended number of buffers according to the requested setup.

MCI_OPEN_PARMS

This structure contains fields for the [MCI_OPEN](#) message.

```
typedef struct _MCI_OPEN_PARMS {  
    HWND      hwndCallback;    /* Window handle. */  
    USHORT    usDeviceID;      /* Device ID. */  
    USHORT    usReserved0;     /* Reserved. */  
    PSZ       pszDeviceType;    /* Device type. */  
    PSZ       pszElementName;   /* Element name. */  
    PSZ       pszAlias;         /* Device alias. */  
} MCI_OPEN_PARMS;  
  
typedef MCI_OPEN_PARMS *PMCI_OPEN_PARMS;
```

MCI_OPEN_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_OPEN_PARMS Field - usDeviceID

usDeviceID ([USHORT](#))
The device ID returned to the user.

MCI_OPEN_PARMS Field - usReserved0

usReserved0 ([USHORT](#))
Reserved.

MCI_OPEN_PARMS Field - pszDeviceType

pszDeviceType ([PSZ](#))
Device type.

MCI_OPEN_PARMS Field - pszElementName

pszElementName ([PSZ](#))
The media element or NULL.

MCI_OPEN_PARMS Field - pszAlias

pszAlias ([PSZ](#))
An optional device alias.

MCI_OVLY_OPEN_PARMS

This structure is the same as [MCI_VID_OPEN_PARMS](#).

```
typedef MCI_VID_OPEN_PARMS MCI_OVLY_OPEN_PARMS;
```

MCI_OVLY_RECT_PARMS

This structure is the same as [MCI_VID_RECT_PARMS](#)

```
typedef MCI_VID_RECT_PARMS MCI_OVLY_RECT_PARMS;
```

MCI_OVLY_SET_PARMS

This structure contains fields for the [MCI_SET](#) message for video overlay devices. This structure is the same as [MCI_SET_PARMS](#).

```
typedef MCI_SET_PARMS MCI_OVLY_SET_PARMS;
```

MCI_OVLY_WINDOW_PARMS

This structure contains fields for the [MCI_WINDOW](#) message for video overlay devices. This structure is the same as [MCI_VID_WINDOW_PARMS](#). When assigning data to the fields in this data structure, set the corresponding Media Control Interface flags in the *ulParam1* parameter of [mciSendCommand](#) to validate the fields.

```
typedef MCI_VID_WINDOW_PARMS MCI_OVLY_WINDOW_PARMS;
```

MCI_PALETTE_PARMS

This structure contains fields for the [MCI_GETIMAGEPALETTE](#) and [MCI_SETIMAGEPALETTE](#) messages.

```
typedef struct _MCI_PALETTE_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    USHORT    usRegisteredMap; /* Registered palette number. */  
    USHORT    usReserved0; /* Reserved. */  
    ULONG     ulPalEntries; /* Palette entries. */  
    PVOID     pPalette; /* RGB entries. */  
} MCI_PALETTE_PARMS;  
  
typedef MCI_PALETTE_PARMS *PMCI_PALETTE_PARMS;
```

MCI_PALETTE_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_PALETTE_PARMS Field - usRegisteredMap

usRegisteredMap ([USHORT](#))
Registered palette number.

MCI_PALETTE_PARMS Field - usReserved0

usReserved0 ([USHORT](#))
Reserved.

MCI_PALETTE_PARMS Field - ulPalEntries

ulPalEntries ([ULONG](#))
Number of RGB entries to be set on an [MCI_SETIMAGEPALETTE](#) message, or number of entries placed in *pPalette* on an [MCI_GETIMAGEPALETTE](#) message.

MCI_PALETTE_PARMS Field - pPalette

pPalette ([PVOID](#))
Array of RGB entries.

MCI_PLAY_PARMS

This structure contains fields for the [MCI_PLAY](#) message.

```
typedef struct _MCI_PLAY_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    ULONG     ulFrom;        /* Starting play position. */  
    ULONG     ulTo;          /* Ending play position. */  
} MCI_PLAY_PARMS;  
  
typedef MCI_PLAY_PARMS *PMCI_PLAY_PARMS;
```

MCI_PLAY_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the [MCI_NOTIFY](#) flag is specified.

MCI_PLAY_PARMS Field - ulFrom

ulFrom ([ULONG](#))
The position to play from.

MCI_PLAY_PARMS Field - ulTo

ulTo ([ULONG](#))
The position to play to.

MCI_POSITION_PARMS

This structure contains fields for the [MCI_SET_POSITION_ADVISE](#) message.

```
typedef struct _MCI_POSITION_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    ULONG      ulUnits;      /* Position change granularity. */  
    USHORT     usUserParm;   /* User parameter. */  
    USHORT     usReserved0;  /* Reserved. */  
    ULONG      ulReserved1;  /* Reserved. */  
} MCI_POSITION_PARMS;  
  
typedef MCI_POSITION_PARMS *PMCI_POSITION_PARMS;
```

MCI_POSITION_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle to be used in returning asynchronous notification. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_POSITION_PARMS Field - ulUnits

ulUnits ([ULONG](#))
Specifies the position-change notification interval in the currently set time format for the device.

MCI_POSITION_PARMS Field - usUserParm

usUserParm ([USHORT](#))

User parameter returned on position-change notification messages ([MM_MCIPOSITIONCHANGE](#)).

MCI_POSITION_PARMS Field - usReserved0

usReserved0 ([USHORT](#))

Reserved.

MCI_POSITION_PARMS Field - ulReserved1

ulReserved1 ([ULONG](#))

Reserved.

MCI_RECORD_PARMS

This structure contains fields for the [MCI_RECORD](#) message.

```
typedef struct _MCI_RECORD_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    ULONG      ulFrom;       /* Beginning record position. */  
    ULONG      ulTo;         /* Ending recording position. */  
} MCI_RECORD_PARMS;
```

```
typedef MCI_RECORD_PARMS *PMCI_RECORD_PARMS;
```

MCI_RECORD_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_RECORD_PARMS Field - ulFrom

ulFrom ([ULONG](#))

The media position at which recording is to begin.

MCI_RECORD_PARMS Field - ulTo

ulTo ([ULONG](#))

The media position at which recording is to end.

MCI_RESTORE_PARMS

This structure contains fields for the [MCI_RESTORE](#) message.

```
typedef struct _MCI_RESTORE_PARMS {
    HWND     hwndCallback; /* Window handle. */
    RECTL     SrcRect;      /* Source rectangle. */
    RECTL     DestRect;     /* Destination rectangle. */
} MCI_RESTORE_PARMS;

typedef MCI_RESTORE_PARMS *PMCI_RESTORE_PARMS;
```

MCI_RESTORE_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_RESTORE_PARMS Field - SrcRect

SrcRect ([RECTL](#))

The source image rectangle of the area to restore.

M-Motion specific: Not used.

MCI_RESTORE_PARMS Field - DestRect

DestRect ([RECTL](#))

The destination window rectangle of the area to restore.

MCI_SAVE_PARMS

This structure contains fields for the [MCI_SAVE](#) message.

```
typedef struct _MCI_SAVE_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    PSZ      pszFileName; /* File name. */  
} MCI_SAVE_PARMS;  
  
typedef MCI_SAVE_PARMS *PMCI_SAVE_PARMS;
```

MCI_SAVE_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_SAVE_PARMS Field - pszFileName

pszFileName ([PSZ](#))

A pointer to a null-terminated buffer that contains a file name.

MCI_SEEK_PARMS

This structure contains fields for the [MCI_SEEK](#) message.

```
typedef struct _MCI_SEEK_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    ULONG     ulTo; /* Target position for seek. */  
} MCI_SEEK_PARMS;  
  
typedef MCI_SEEK_PARMS *PMCI_SEEK_PARMS;
```

MCI_SEEK_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_SEEK_PARMS Field - ulTo

ulTo ([ULONG](#))

The media position that is the target of the seek operation.

MCI_SEQ_SET_PARMS

This structure contains fields for the [MCI_SET](#) message used with MIDI sequencer devices.

```
typedef struct _MCI_SEQ_SET_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG     ulTimeFormat; /* Device time format. */
    ULONG     ulSpeedFormat; /* Speed format of the device. */
    ULONG     ulAudio; /* Channel number. */
    ULONG     ulLevel; /* Volume as percentage. */
    ULONG     ulOver; /* Delay time. */
    ULONG     ulItem; /* Item field. */
    ULONG     ulValue; /* Item value. */
    ULONG     ulTempo; /* Tempo. */
    ULONG     ulPort; /* Output port. */
    ULONG     ulSlave; /* Not used. */
    ULONG     ulMaster; /* Not used. */
    ULONG     ulOffset; /* Data offset. */
} MCI_SEQ_SET_PARMS;

typedef MCI_SEQ_SET_PARMS *PMCI_SEQ_SET_PARMS;
```

MCI_SEQ_SET_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_SEQ_SET_PARMS Field - ulTimeFormat

ulTimeFormat ([ULONG](#))

The time format to be used by the device.

MCI_SEQ_SET_PARMS Field - ulSpeedFormat

ulSpeedFormat (ULONG)
The speed format to be used by this device.

MCI_SEQ_SET_PARMS Field - ulAudio

ulAudio (ULONG)
Channel number for the following operations:

- MCI_SET_AUDIO_LEFT
- MCI_SET_AUDIO_RIGHT
- MCI_SET_AUDIO_ALL

MCI_SEQ_SET_PARMS Field - ulLevel

ulLevel (ULONG)
Audio volume level as a percentage of maximum.

MCI_SEQ_SET_PARMS Field - ulOver

ulOver (ULONG)
Delay time for vectored change, in milliseconds.

MCI_SEQ_SET_PARMS Field - ullItem

ullItem (ULONG)
Field for set item flags such as tempo, SMPTE offset, and so forth.

MCI_SEQ_SET_PARMS Field - ulValue

ulValue (ULONG)

Value associated with item flag.

MCI_SEQ_SET_PARMS Field - ulTempo

ulTempo (ULONG)
Tempo.

MCI_SEQ_SET_PARMS Field - ulPort

ulPort (ULONG)
Output port.

MCI_SEQ_SET_PARMS Field - ulSlave

ulSlave (ULONG)
Not used.

MCI_SEQ_SET_PARMS Field - ulMaster

ulMaster (ULONG)
Not used.

MCI_SEQ_SET_PARMS Field - ulOffset

ulOffset (ULONG)
Data offset.

MCI_SET_PARMS

This structure contains fields for the [MCI_SET](#) message for devices that do not have device-specific extensions.

```
typedef struct _MCI_SET_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG     ulTimeFormat; /* Time format. */
    ULONG     ulSpeedFormat; /* Speed format. */
    ULONG     ulAudio; /* Channel number. */
    ULONG     ulLevel; /* Volume level. */
    ULONG     ulOver; /* Delay time. */
    ULONG     ulItem; /* Item field. */
    ULONG     ulValue; /* Value for item flag. */
} MCI_SET_PARMS;

typedef MCI_SET_PARMS *PMCI_SET_PARMS;
```

MCI_SET_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_SET_PARMS Field - ulTimeFormat

ulTimeFormat (ULONG)

The time format to be used by the device.

MCI_SET_PARMS Field - ulSpeedFormat

ulSpeedFormat (ULONG)

The speed format to be used by the device.

Note: This has no effect for video overlay devices.

MCI_SET_PARMS Field - ulAudio

ulAudio (ULONG)

Channel number for the following operations:

- MCI_SET_AUDIO_LEFT
- MCI_SET_AUDIO_RIGHT
- MCI_SET_AUDIO_ALL

MCI_SET_PARMS Field - ulLevel

ulLevel ([ULONG](#))

Audio volume level as a percentage of maximum.

Note: This has no effect for video overlay devices.

MCI_SET_PARMS Field - ulOver

ulOver ([ULONG](#))

Delay time for vectored change, in milliseconds.

MCI_SET_PARMS Field - ullItem

ullItem ([ULONG](#))

Field for set item flags for such items as file format, image pelformat, brightness, and so forth.

MCI_SET_PARMS Field - ulValue

ulValue ([ULONG](#))

Value associated with item flag that sets items such as image pelformat and image file format. The *ulValue* is interpreted as a FOURCC value.

MCI_STATUS_PARMS

This structure contains fields for the [MCI_STATUS](#) message.

```
typedef struct _MCI_STATUS_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG     ulReturn;      /* Return information. */
    ULONG     ulItem;        /* Status item. */
    ULONG     ulValue;       /* Status value. */
} MCI_STATUS_PARMS;

typedef MCI_STATUS_PARMS *PMCI_STATUS_PARMS;
```

MCI_STATUS_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle cast to a ULONG to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_STATUS_PARMS Field - ulReturn

ulReturn ([ULONG](#))

Contains the status information upon return. High-order word could be value or other message.

The format of this value is defined by the high-order word of the value returned by the [mciSendCommand](#). This value is used by the [mciSendString](#) to determine how to convert the *ulReturn* value to string form. See the MMDRVOS2.H header file for possible format values.

MCI_STATUS_PARMS Field - ullItem

ullItem ([ULONG](#))

The status item to query.

MCI_STATUS_PARMS Field - ulValue

ulValue ([ULONG](#))

The status value to extend the status structure.

MCI_STEP_PARMS

This structure contains fields for the [MCI_STEP](#) message.

```
typedef struct _MCI_STEP_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG     ulStep;        /* Step increment. */
} MCI_STEP_PARMS;

typedef MCI_STEP_PARMS *PMCI_STEP_PARMS;
```

MCI_STEP_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_STEP_PARMS Field - ulStep

ulStep ([ULONG](#))

The increment for which the step is specified in the current time format.

MCI_SYNC_OFFSET_PARMS

This structure contains fields for the [MCI_SET_SYNC_OFFSET](#) message.

```
typedef struct _MCI_SYNC_OFFSET_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG      ulOffset;     /* Media offset of current time format. */
} MCI_SYNC_OFFSET_PARMS;

typedef MCI_SYNC_OFFSET_PARMS *PMCI_SYNC_OFFSET_PARMS;
```

MCI_SYNC_OFFSET_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_SYNC_OFFSET_PARMS Field - ulOffset

ulOffset ([ULONG](#))

The device media-position offset in the currently specified time format.

MCI_SYSINFO_ALIAS

This structure contains fields for the [MCI_SYSINFO](#) message for setting and querying an alias associated with a particular device.

```
typedef struct _MCI_SYSINFO_ALIAS {
```

```
    CHAR        szInstallName[MAX_DEVICE_NAME]; /* Device installation name. */
    CHAR        szAliasName[MAX_ALIAS_NAME]; /* Alias name. */
} MCI_SYSINFO_ALIAS;

typedef MCI_SYSINFO_ALIAS *PMCI_SYSINFO_ALIAS;
```

MCI_SYSINFO_ALIAS Field - szInstallName[MAX_DEVICE_NAME]

szInstallName[MAX_DEVICE_NAME] ([CHAR](#))
Device installation name.

MCI_SYSINFO_ALIAS Field - szAliasName[MAX_ALIAS_NAME]

szAliasName[MAX_ALIAS_NAME] ([CHAR](#))
Alias name.

MCI_SYSINFO_CONPARAMS

This structure contains fields for the [MCI_SYSINFO](#) message.

```
typedef struct _MCI_SYSINFO_CONPARAMS {
    CHAR        szInstallName[MAX_DEVICE_NAME]; /* Device install name. */
    USHORT      usNumConnectors; /* Number of device connectors. */
    CONNECT     ConnectorList[MAX_CONNECTORS]; /* Connector list array. */
} MCI_SYSINFO_CONPARAMS;

typedef MCI_SYSINFO_CONPARAMS *PMCI_SYSINFO_CONPARAMS;
```

MCI_SYSINFO_CONPARAMS Field - szInstallName[MAX_DEVICE_NAME]

szInstallName[MAX_DEVICE_NAME] ([CHAR](#))
Device install name.

MCI_SYSINFO_CONPARAMS Field - usNumConnectors

usNumConnectors ([USHORT](#))
Number of device connectors.

MCI_SYSINFO_CONPARAMS Field - ConnectorList[MAX_CONNECTORS]

ConnectorList[MAX_CONNECTORS] ([CONNECT](#))
Pointer to array of device connectors. See [CONNECT](#).

MCI_SYSINFO_DEFAULTDEVICE

This structure contains fields for the [MCI_SYSINFO](#) message for setting and querying a default associated with a particular device type.

```
typedef struct _MCI_SYSINFO_DEFAULTDEVICE {  
    CHAR          szInstallName[MAX_DEVICE_NAME]; /* Install name. */  
    USHORT        usDeviceType; /* Device type number. */  
} MCI_SYSINFO_DEFAULTDEVICE;  
  
typedef MCI_SYSINFO_DEFAULTDEVICE *PMCI_SYSINFO_DEFAULTDEVICE;
```

MCI_SYSINFO_DEFAULTDEVICE Field - szInstallName[MAX_DEVICE_NAME]

szInstallName[MAX_DEVICE_NAME] ([CHAR](#))
Device install name.

MCI_SYSINFO_DEFAULTDEVICE Field - usDeviceType

usDeviceType ([USHORT](#))
Device type number.

MCI_SYSINFO_DEVPARAMS

This structure contains fields for the [MCI_SYSINFO](#) message for setting and querying device-specific fields.

```
typedef struct _MCI_SYSINFO_DEVPARAMS {  
    CHAR        szInstallName[MAX_DEVICE_NAME]; /* Install Name. */  
    CHAR        szDevParams[MAX_DEV_PARAMS]; /* Device-specific fields. */  
} MCI_SYSINFO_DEVPARAMS;  
  
typedef MCI_SYSINFO_DEVPARAMS *PMCI_SYSINFO_DEVPARAMS;
```

MCI_SYSINFO_DEVPARAMS Field - szInstallName[MAX_DEVICE_NAME]

szInstallName[MAX_DEVICE_NAME] ([CHAR](#))
Device install name.

MCI_SYSINFO_DEVPARAMS Field - szDevParams[MAX_DEV_PARAMS]

szDevParams[MAX_DEV_PARAMS] ([CHAR](#))
Device-specific fields.

MCI_SYSINFO_EXTENSION

This structure contains fields for the [MCI_SYSINFO](#) message for setting and querying file extensions associated with a particular device.

```
typedef struct _MCI_SYSINFO_EXTENSION {  
    CHAR        szInstallName[MAX_DEVICE_NAME]; /* Device install name. */  
    USHORT      usNumExtensions; /* Number of extensions. */  
    CHAR        szExtension[MAX_EXTENSIONS][MAX_EXTENSION_NAME]; /* Extension name array. */  
} MCI_SYSINFO_EXTENSION;  
  
typedef MCI_SYSINFO_EXTENSION *PMCI_SYSINFO_EXTENSION;
```

MCI_SYSINFO_EXTENSION Field - szInstallName[MAX_DEVICE_NAME]

szInstallName[MAX_DEVICE_NAME] ([CHAR](#))
Device install name.

MCI_SYSINFO_EXTENSION Field - usNumExtensions

usNumExtensions (USHORT)
Number of extensions.

MCI_SYSINFO_EXTENSION Field - szExtension[MAX_EXTENSIONS][MAX_EXTENSION_NAME]

szExtension[MAX_EXTENSIONS][MAX_EXTENSION_NAME] (CHAR)
Extension name array.

MCI_SYSINFO_LOGDEVICE

This structure contains fields for the [MCI_SYSINFO](#) message for setting and querying logical devices.

```
typedef struct _MCI_SYSINFO_LOGDEVICE {
    CHAR        szInstallName[MAX_DEVICE_NAME];           /* Device install name. */
    USHORT      usDeviceType;                             /* Device type name. */
    ULONG       ulDeviceFlag;                             /* Device controllable flag. */
    CHAR        szVersionNumber[MAX_VERSION_NUMBER];      /* INI file version number. */
    CHAR        szProductInfo[MAX_PRODINFO];              /* Textual product description. */
    CHAR        szMCDDriver[MAX_DEVICE_NAME];             /* Driver DLL name. */
    CHAR        szVSDDriver[MAX_DEVICE_NAME];             /* VSD DLL name. */
    CHAR        szPDDName[MAX_PDD_NAME];                 /* PDD name. */
    CHAR        szMCDTable[MAX_DEVICE_NAME];             /* Device type command table. */
    CHAR        szVSDTable[MAX_DEVICE_NAME];             /* Device specific command table. */
    USHORT      usShareType;                              /* Device sharing mode. */
    CHAR        szResourceName[MAX_DEVICE_NAME];         /* Resource name. */
    USHORT      usResourceUnits;                         /* Resource units available. */
    USHORT      usResourceClasses;                       /* Number of resource classes. */
    USHORT      ausClassArray[MAX_CLASSES];              /* Maximum resource units per class. */
    USHORT      ausValidClassArray[MAX_CLASSES][MAX_CLASSES]; /* Valid class combinations. */
} MCI_SYSINFO_LOGDEVICE;

typedef MCI_SYSINFO_LOGDEVICE *PMCI_SYSINFO_LOGDEVICE;
```

MCI_SYSINFO_LOGDEVICE Field - szInstallName[MAX_DEVICE_NAME]

szInstallName[MAX_DEVICE_NAME] (CHAR)
Device install name.

MCI_SYSINFO_LOGDEVICE Field - usDeviceType

usDeviceType ([USHORT](#))
Device type number.

MCI_SYSINFO_LOGDEVICE Field - ulDeviceFlag

ulDeviceFlag ([ULONG](#))
Flag indicating whether device is controllable or not (MCI_SYSINFO_DEV_CONTROLLABLE and MCI_SYSINFO_DEV_NONCONTROLLABLE)

The following styles can be used.

MCI_SYSINFO_DEVICESETTINGS
Indicates the MCD has custom device settings pages.

MCI_SYSINFO_DEV_CONTROLLABLE
If a device is controllable, it usually accepts a PLAY command.

MCI_SYSINFO_DEV_NONCONTROLLABLE
Examples of non-controllable devices are speakers, headphones, microphone, and amp-mixer devices.

MCI_SYSINFO_LOGDEVICE Field - szVersionNumber[MAX_VERSION_NUMBER]

szVersionNumber[MAX_VERSION_NUMBER] ([CHAR](#))
INI file version number.

MCI_SYSINFO_LOGDEVICE Field - szProductInfo[MAX_PRODINFO]

szProductInfo[MAX_PRODINFO] ([CHAR](#))
Textual product description.

MCI_SYSINFO_LOGDEVICE Field -

szMCDDriver[MAX_DEVICE_NAME]

szMCDDriver[MAX_DEVICE_NAME] (CHAR)
Media control interface driver DLL name.

MCI_SYSINFO_LOGDEVICE Field - szVSDDriver[MAX_DEVICE_NAME]

szVSDDriver[MAX_DEVICE_NAME] (CHAR)
Vendor-specific driver DLL name.

MCI_SYSINFO_LOGDEVICE Field - szPDDName[MAX_PDD_NAME]

szPDDName[MAX_PDD_NAME] (CHAR)
Device PDD name. The device driver name must not be more than eight characters (excluding the file extension).

MCI_SYSINFO_LOGDEVICE Field - szMCDTable[MAX_DEVICE_NAME]

szMCDTable[MAX_DEVICE_NAME] (CHAR)
Device type command table.

MCI_SYSINFO_LOGDEVICE Field - szVSDTable[MAX_DEVICE_NAME]

szVSDTable[MAX_DEVICE_NAME] (CHAR)
Device-specific command table.

MCI_SYSINFO_LOGDEVICE Field - usShareType

usShareType (USHORT)
Device sharing mode.

MCI_SYSINFO_LOGDEVICE Field - szResourceName[MAX_DEVICE_NAME

szResourceName[MAX_DEVICE_NAME] (CHAR)
Resource name.

MCI_SYSINFO_LOGDEVICE Field - usResourceUnits

usResourceUnits (USHORT)
Total resource units available for this device.

MCI_SYSINFO_LOGDEVICE Field - usResourceClasses

usResourceClasses (USHORT)
Number of resource classes for this device.

MCI_SYSINFO_LOGDEVICE Field - ausClassArray[MAX_CLASSES]

ausClassArray[MAX_CLASSES] (USHORT)
Maximum number of resource units for each class.

MCI_SYSINFO_LOGDEVICE Field - ausValidClassArray[MAX_CLASSES][MAX_CLASSES]

ausValidClassArray[MAX_CLASSES][MAX_CLASSES] (USHORT)
Valid class combinations. A 1 in the matrix indicates a valid combination.

MCI_SYSINFO_PARMS

This structure contains fields for the [MCI_SYSINFO](#) message.

```
typedef struct _MCI_SYSINFO_PARMS {
    ULONG      ulDummyCallback; /* Notify not valid. */
    PSZ        pszReturn;      /* Pointer to application-supplied buffer. */
    ULONG      ulRetSize;      /* Size of return string buffer. */
    ULONG      ulNumber;       /* Device ordinal. */
    USHORT     usDeviceType;    /* Device type. */
    USHORT     usReserved0;     /* Reserved. */
    ULONG      ulItem;          /* Indicates extended function. */
    PVOID      pSysInfoParam;   /* Pointer to specific info. */
} MCI_SYSINFO_PARMS;

typedef MCI_SYSINFO_PARMS *PMCI_SYSINFO_PARMS;
```

MCI_SYSINFO_PARMS Field - ulDummyCallback

ulDummyCallback ([ULONG](#))
Reserved. Notify not valid for [MCI_SYSINFO](#).

MCI_SYSINFO_PARMS Field - pszReturn

pszReturn ([PSZ](#))
A pointer to an application-supplied buffer for the return string.

MCI_SYSINFO_PARMS Field - ulRetSize

ulRetSize ([ULONG](#))
The size, in bytes, of the buffer for the return string.

MCI_SYSINFO_PARMS Field - ulNumber

ulNumber ([ULONG](#))
The device number or ordinal. This number ranges from 1 to the number of devices of that type.

MCI_SYSINFO_PARMS Field - usDeviceType

usDeviceType (USHORT)
The type of device.

MCI_SYSINFO_PARMS Field - usReserved0

usReserved0 (USHORT)
Reserved.

MCI_SYSINFO_PARMS Field - ullItem

ullItem (ULONG)
Used to indicate the MCI_SYSINFO extended function to perform.

MCI_SYSINFO_PARMS Field - pSysInfoParm

pSysInfoParm (PVOID)
Pointer to specific information or a specific data structure according to the value specified for the *ullItem* field.

MCI_SYSINFO_QUERY_NAME

This structure contains fields for the **MCI_SYSINFO** message for querying the names associated with a particular device.

```
typedef struct _MCI_SYSINFO_QUERY_NAME {  
    CHAR        szInstallName[MAX_DEVICE_NAME]; /* Device install name. */  
    CHAR        szLogicalName[MAX_DEVICE_NAME]; /* Logical device name. */  
    CHAR        szAliasName[MAX_ALIAS_NAME]; /* Alias name. */  
    USHORT      usDeviceType; /* Device type number. */  
    USHORT      usDeviceOrd; /* Device type ordinal. */  
} MCI_SYSINFO_QUERY_NAME;  
  
typedef MCI_SYSINFO_QUERY_NAME *PMCI_SYSINFO_QUERY_NAME;
```

MCI_SYSINFO_QUERY_NAME Field - szInstallName[MAX_DEVICE_NAME]

szInstallName[MAX_DEVICE_NAME] (CHAR)
Device install name.

MCI_SYSINFO_QUERY_NAME Field - szLogicalName[MAX_DEVICE_NAME]

szLogicalName[MAX_DEVICE_NAME] (CHAR)
Logical device name.

MCI_SYSINFO_QUERY_NAME Field - szAliasName[MAX_ALIAS_NAME]

szAliasName[MAX_ALIAS_NAME] (CHAR)
Alias name.

MCI_SYSINFO_QUERY_NAME Field - usDeviceType

usDeviceType (USHORT)
Device type number.

MCI_SYSINFO_QUERY_NAME Field - usDeviceOrd

usDeviceOrd (USHORT)
Device type ordinal.

MCI_SYSINFO_TYPES

This structure contains fields for the [MCI_SYSINFO](#) message for setting and querying extended attributes associated with a particular device.

```
typedef struct _MCI_SYSINFO_TYPES {
    CHAR    szInstallName[MAX_DEVICE_NAME]; /* Install name. */
    CHAR    szTypes[MAX_TYPEBUFFER+1]; /* Extended-type array. */
} MCI_SYSINFO_TYPES;

typedef MCI_SYSINFO_TYPES *PMCI_SYSINFO_TYPES;
```

MCI_SYSINFO_TYPES Field - szInstallName[MAX_DEVICE_NAME]

szInstallName[MAX_DEVICE_NAME] ([CHAR](#))
Device install name.

MCI_SYSINFO_TYPES Field - szTypes[MAX_TYPEBUFFER+1]

szTypes[MAX_TYPEBUFFER+1] ([CHAR](#))
Extended-type array. The attributes are separated by a comma and the list is null-terminated.

MCI_TOC_PARMS

This structure contains fields for the [MCI_GETTOC](#) message.

```
typedef struct _MCI_TOC_PARMS {
    HWND    hwndCallback; /* Window handle. */
    PTOCREC pBuf; /* Pointer to an array. */
    ULONG   ulBufSize; /* Size of TOC records array. */
} MCI_TOC_PARMS;

typedef MCI_TOC_PARMS *PMCI_TOC_PARMS;
```

MCI_TOC_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_TOC_PARMS Field - pBuf

pBuf ([PTOCREC](#))

Pointer to array of [MCI_TOC_REC](#) structures that are to be filled in by the media control interface device.

MCI_TOC_PARMS Field - ulBufSize

ulBufSize ([ULONG](#))

Size of the array of [MCI_TOC_REC](#) structures.

MCI_TOC_REC

This structure contains fields that make up the TOC record.

```
typedef struct _MCI_TOC_REC {
    BYTE      TrackNum;      /* Track number. */
    ULONG     ulStartAddr;   /* Starting address in MMTIME. */
    ULONG     ulEndAddr;     /* Ending address in MMTIME. */
    BYTE      Control;       /* Track control info. */
    USHORT    usCountry;     /* Country. */
    ULONG     ulOwner;       /* Owner. */
    ULONG     ulSerialNum;   /* Serial number. */
} MCI_TOC_REC;

typedef MCI_TOC_REC *PTOCREC;
```

MCI_TOC_REC Field - TrackNum

TrackNum ([BYTE](#))

Returned track number.

MCI_TOC_REC Field - ulStartAddr

ulStartAddr ([ULONG](#))

Starting address of the track in [MMTIME](#) format.

MCI_TOC_REC Field - ulEndAddr

ulEndAddr (ULONG)
Ending address of the track in MMTIME format.

MCI_TOC_REC Field - Control

Control (BYTE)
Track control information.

MCI_TOC_REC Field - usCountry

usCountry (USHORT)
Country.

MCI_TOC_REC Field - ulOwner

ulOwner (ULONG)
Owner.

MCI_TOC_REC Field - ulSerialNum

ulSerialNum (ULONG)
Serial number.

MCI_VD_PLAY_PARMS

This structure contains fields for the MCI_PLAY message for videodiscs.

```
typedef struct _MCI_VD_PLAY_PARMS {
    HWND      hwndCallback; /* Window handle. */
    ULONG      ulFrom;       /* Starting play position. */
}
```



```

    ULONG    ulTo;           /* Ending play position. */
    ULONG    ulFactor;       /* Current speed format factor. */
} MCI_VD_PLAY_PARMS;

typedef MCI_VD_PLAY_PARMS *PMCI_VD_PLAY_PARMS;

```

MCI_VD_PLAY_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_VD_PLAY_PARMS Field - ulFrom

ulFrom (ULONG)

The position to play from.

MCI_VD_PLAY_PARMS Field - ulTo

ulTo (ULONG)

The position to play to.

MCI_VD_PLAY_PARMS Field - ulFactor

ulFactor (ULONG)

The speed factor for playing in the current speed format.

MCI_VD_SET_PARMS

This structure contains fields for the [MCI_SET](#) message for videodiscs.

```

typedef struct _MCI_VD_SET_PARMS {
    HWND    hwndCallback;    /* Window handle. */
    ULONG    ulTimeFormat;    /* Device time format. */
    ULONG    ulSpeedFormat;   /* Device speed format. */
    ULONG    ulAudio;         /* Channel number. */
    ULONG    ulLevel;         /* Volume as percent. */
}

```

```
ULONG      ulOver;          /* Delay time. */
ULONG      ulItem;          /* Item field. */
ULONG      ulValue;         /* Value associated with item flag. */
ULONG      ulChannel;       /* The videodisc channel. */
} MCI_VD_SET_PARMS;

typedef MCI_VD_SET_PARMS *PMCI_VD_SET_PARMS;
```

MCI_VD_SET_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle cast to a ULONG to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_VD_SET_PARMS Field - ulTimeFormat

ulTimeFormat (ULONG)

The time format to be used by the device.

MCI_VD_SET_PARMS Field - ulSpeedFormat

ulSpeedFormat (ULONG)

The speed format to be used by the device.

MCI_VD_SET_PARMS Field - ulAudio

ulAudio (ULONG)

Channel number for the following operations:

- MCI_SET_AUDIO_LEFT
- MCI_SET_AUDIO_RIGHT
- MCI_SET_AUDIO_ALL

MCI_VD_SET_PARMS Field - ulLevel

ulLevel (ULONG)

Audio volume level as a percentage of maximum.

MCI_VD_SET_PARMS Field - ulOver

ulOver ([ULONG](#))

Delay time for vectored change, in milliseconds.

MCI_VD_SET_PARMS Field - ullItem

ullItem ([ULONG](#))

Field for set item flags such as channel, display, and so forth.

MCI_VD_SET_PARMS Field - ulValue

ulValue ([ULONG](#))

Value associated with item flag.

MCI_VD_SET_PARMS Field - ulChannel

ulChannel ([ULONG](#))

The videodisc channel.

MCI_VID_OPEN_PARMS

This structure contains fields for the [MCI_OPEN](#) message for digital video devices.

```
typedef struct _MCI_VID_OPEN_PARMS {
    HWND      hwndCallback; /* Window handle. */
    USHORT    usDeviceID;    /* Device ID. */
    USHORT    usReserved0;   /* Reserved. */
    PSZ       pszDeviceType; /* Device type. */
    PSZ       pszElementName; /* Media element name or NULL. */
    PSZ       pszAlias;      /* Optional device alias. */
    HWND      hwndParent;    /* Parent window handle. */
} MCI_VID_OPEN_PARMS;

typedef MCI_VID_OPEN_PARMS *PMCI_VID_OPEN_PARMS;
```

MCI_VID_OPEN_PARMS Field - hwndCallback

hwndCallback ([HWND](#))
A window handle used for returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_VID_OPEN_PARMS Field - usDeviceID

usDeviceID ([USHORT](#))
The device ID returned to the user.

MCI_VID_OPEN_PARMS Field - usReserved0

usReserved0 ([USHORT](#))
Reserved.

MCI_VID_OPEN_PARMS Field - pszDeviceType

pszDeviceType ([PSZ](#))
The type of the device.

MCI_VID_OPEN_PARMS Field - pszElementName

pszElementName ([PSZ](#))
The media element name (usually a path name) or NULL.

MCI_VID_OPEN_PARMS Field - pszAlias

pszAlias ([PSZ](#))

An optional device alias. For no alias, use NULL.

MCI_VID_OPEN_PARMS Field - hwndParent

hwndParent ([HWND](#))

The handle to use as the window parent.

MCI_VID_RECT_PARMS

This structure contains fields for the [MCI_PUT](#) and [MCI_WHERE](#) messages for digital video devices.

```
typedef struct _MCI_VID_RECT_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    RECTL     rc;           /* Specified rectangle array. */  
} MCI_VID_RECT_PARMS;  
  
typedef MCI_VID_RECT_PARMS *PMCI_VID_RECT_PARMS;
```

MCI_VID_RECT_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle used in returning asynchronous notification. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_VID_RECT_PARMS Field - rc

rc ([RECTL](#))

Specifies or returns a rectangle array specifying the offset and size of a rectangle, depending on the media control interface command.

MCI_VID_WINDOW_PARMS

This structure contains fields for the [MCI_WINDOW](#) message for digital video devices.

```
typedef struct _MCI_VID_WINDOW_PARMS {  
    HWND      hwndCallback; /* Window handle. */  
    HWND      hwndDest;     /* Handle for destination client window. */  
    USHORT    usCmdShow;     /* Window display specification. */  
}
```

```
USHORT    usReserved1;    /* Reserved. */
PSZ       pszText;        /* Window caption test. */
PSZ       pszAlias;       /* Display window alias. */
} MCI_VID_WINDOW_PARMS;

typedef MCI_VID_WINDOW_PARMS *PMCI_VID_WINDOW_PARMS;
```

MCI_VID_WINDOW_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_VID_WINDOW_PARMS Field - hwndDest

hwndDest (HWND)

A handle to the client window used for the destination of the digital video.

MCI_VID_WINDOW_PARMS Field - usCmdShow

usCmdShow (USHORT)

Specifies how the window is displayed.

MCI_VID_WINDOW_PARMS Field - usReserved1

usReserved1 (USHORT)

Reserved.

MCI_VID_WINDOW_PARMS Field - pszText

pszText (PSZ)

The text to use as the window caption.

MCI_VID_WINDOW_PARMS Field - pszAlias

pszAlias (PSZ)

The window alias for the display window.

MCI_WAVE_GETDEVCAPS_PARMS

This structure contains parameters for the [MCI_GETDEVCAPS](#) message.

```
typedef struct _MCI_WAVE_GETDEVCAPS_PARMS {
    HWND      hwndCallback;    /* Window handle. */
    ULONG      ulReturn;        /* Return field. */
    ULONG      ulItem;          /* Item to query. */
    USHORT     usMessage;       /* Message to query. */
    USHORT     usReserved0;     /* Reserved. */
    ULONG      ulLength;        /* Length of structure. */
    ULONG      ulBitsPerSample; /* Bits per sample. */
    ULONG      ulFormatTag;     /* Format tag. */
    ULONG      ulSamplesPerSec; /* Sampling rate. */
    ULONG      ulChannels;      /* Number of channels. */
    ULONG      ulFormatMode;    /* MCI_PLAY or MCI_RECORD. */
} MCI_WAVE_GETDEVCAPS_PARMS;

typedef MCI_WAVE_GETDEVCAPS_PARMS *PMCI_WAVE_GETDEVCAPS_PARMS;
```

MCI_WAVE_GETDEVCAPS_PARMS Field - hwndCallback

hwndCallback (HWND)

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_WAVE_GETDEVCAPS_PARMS Field - ulReturn

ulReturn (ULONG)

Return field.

MCI_WAVE_GETDEVCAPS_PARMS Field - ullItem

ullItem (ULONG)

Item field for [MCI_GETDEVCAPS](#) to query.

MCI_WAVE_GETDEVCAPS_PARMS Field - usMessage

usMessage (USHORT)
Field holds media control interface message to query.

MCI_WAVE_GETDEVCAPS_PARMS Field - usReserved0

usReserved0 (USHORT)
Reserved.

MCI_WAVE_GETDEVCAPS_PARMS Field - ulLength

ulLength (ULONG)
Length of structure.

MCI_WAVE_GETDEVCAPS_PARMS Field - ulBitsPerSample

ulBitsPerSample (ULONG)
Number of bits per sample in a waveaudio sample. Typically 8 or 16.

MCI_WAVE_GETDEVCAPS_PARMS Field - ulFormatTag

ulFormatTag (ULONG)
Format tag.

MCI_WAVE_GETDEVCAPS_PARMS Field - ulSamplesPerSec

ulSamplesPerSec (ULONG)

Samples per second rate of waveform audio.

MCI_WAVE_GETDEVCAPS_PARMS Field - ulChannels

ulChannels ([ULONG](#))

Number of channels. Typically 1 or 2.

MCI_WAVE_GETDEVCAPS_PARMS Field - ulFormatMode

ulFormatMode ([ULONG](#))

The format mode, either [MCI_PLAY](#) or [MCI_RECORD](#).

MCI_WAVE_SET_PARMS

This structure contains fields for the [MCI_SET](#) message for waveform devices.

```
typedef struct _MCI_WAVE_SET_PARMS {
    HWND      hwndCallback;    /* Window handle. */
    ULONG      ulTimeFormat;    /* Device time format. */
    ULONG      ulSpeedFormat;   /* Device speed format. */
    ULONG      ulAudio;         /* Channel - left/right/all. */
    ULONG      ulLevel;         /* Volume as percent. */
    ULONG      ulOver;          /* Delay time. */
    ULONG      ulItem;          /* Item field. */
    ULONG      ulValue;         /* Item flag value. */
    USHORT     usInput;         /* Input channel. */
    USHORT     usReserved0;      /* Reserved. */
    USHORT     usOutput;        /* Output channel. */
    USHORT     usReserved1;      /* Reserved. */
    USHORT     usFormatTag;      /* Format tag. */
    USHORT     usReserved2;      /* Reserved. */
    USHORT     usChannels;       /* Mono=1 Stereo=2. */
    USHORT     usReserved3;      /* Reserved. */
    ULONG      ulSamplesPerSec;  /* Rate used by waveform. */
    ULONG      ulAvgBytesPerSec; /* Data rate (BPS). */
    USHORT     usBlockAlign;     /* Data alignment. */
    USHORT     usReserved4;      /* Reserved. */
    USHORT     usBitsPerSample;  /* Number of bits per sample. */
    USHORT     usReserved5;      /* Reserved. */
} MCI_WAVE_SET_PARMS;
```

```
typedef MCI_WAVE_SET_PARMS *PMCI_WAVE_SET_PARMS;
```

MCI_WAVE_SET_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

A window handle to be used in returning asynchronous notification messages. This parameter must be specified if the MCI_NOTIFY flag is specified.

MCI_WAVE_SET_PARMS Field - ulTimeFormat

ulTimeFormat (ULONG)

The time format to be used by the device.

MCI_WAVE_SET_PARMS Field - ulSpeedFormat

ulSpeedFormat (ULONG)

The speed format to be used by the device.

MCI_WAVE_SET_PARMS Field - ulAudio

ulAudio (ULONG)

Channel set for the following operations:

- MCI_SET_AUDIO_LEFT
 - MCI_SET_AUDIO_RIGHT
 - MCI_SET_AUDIO_ALL
-

MCI_WAVE_SET_PARMS Field - ulLevel

ulLevel (ULONG)

Audio volume level as a percentage of maximum.

MCI_WAVE_SET_PARMS Field - ulOver

ulOver (ULONG)

Delay time for vectored change in milliseconds.

MCI_WAVE_SET_PARMS Field - ullItem

ulItem (ULONG)
Item field for set item flags.

MCI_WAVE_SET_PARMS Field - ulValue

ulValue (ULONG)
Value associated with item flag.

MCI_WAVE_SET_PARMS Field - usInput

usInput (USHORT)
The channel used for input.

MCI_WAVE_SET_PARMS Field - usReserved0

usReserved0 (USHORT)
Reserved.

MCI_WAVE_SET_PARMS Field - usOutput

usOutput (USHORT)
The channel used for output.

MCI_WAVE_SET_PARMS Field - usReserved1

usReserved1 (USHORT)
Reserved.

MCI_WAVE_SET_PARMS Field - usFormatTag

usFormatTag (USHORT)
Specifies the interpretation of the waveform data.

MCI_WAVE_SET_PARMS Field - usReserved2

usReserved2 (USHORT)
Reserved.

MCI_WAVE_SET_PARMS Field - usChannels

usChannels (USHORT)
Specifies mono (1) or stereo (2).

MCI_WAVE_SET_PARMS Field - usReserved3

usReserved3 (USHORT)
Reserved.

MCI_WAVE_SET_PARMS Field - ulSamplesPerSec

ulSamplesPerSec (ULONG)
The samples per second used for the waveform.

MCI_WAVE_SET_PARMS Field - ulAvgBytesPerSec

ulAvgBytesPerSec (ULONG)
The average data rate in bytes per second.

MCI_WAVE_SET_PARMS Field - usBlockAlign

usBlockAlign ([USHORT](#))
The block alignment of the data.

MCI_WAVE_SET_PARMS Field - usReserved4

usReserved4 ([USHORT](#))
Reserved.

MCI_WAVE_SET_PARMS Field - usBitsPerSample

usBitsPerSample ([USHORT](#))
The number of bits per sample.

MCI_WAVE_SET_PARMS Field - usReserved5

usReserved5 ([USHORT](#))
Reserved.

MESSAGE

This structure contains fields for the [MIDISendMessages](#) function.

```
typedef struct {
    ULONG    ulSourceInstance; /* Source instance. */
    ULONG    ulTime;          /* Time for sending message. */
    ULONG    ulTrack;         /* Reserved. */
    union {
        ULONG    ulMessage;    /* MIDI message. */
        struct {
            BYTE    bStatus;    /* First byte of message. */
            BYTE    abData[3];  /* The rest of the message. */
        } bytes;
        BYTE    abData[4];      /* MIDI message. */
    } msg; /* MIDI message. */
} MESSAGE;

typedef MESSAGE *PMESSAGE;
```

MESSAGE Field - ulSourceInstance

ulSourceInstance (ULONG)
Source instance.

MESSAGE Field - ulTime

ulTime (ULONG)
The time the message is to be (or was) sent.

MESSAGE Field - ulTrack

ulTrack (ULONG)
This field is reserved and should be set to 0.

MESSAGE Field - ulMessage

ulMessage (ULONG)
An alias for the bytes structure allowing access to the MIDI message as a single ULONG.

MESSAGE Field - bStatus

bStatus (BYTE)
First byte of message.

MESSAGE Field - abData[3]

abData[3] (BYTE)
The rest of the message.

MESSAGE Field - abData[4]

abData[4] ([BYTE](#))

An alias for the bytes structure allowing access to the MIDI message as a single array.

MIDICLASSINFO

This structure contains fields for the [MIDIQueryClassList](#) function.

```
typedef struct _MIDICLASSINFO {
    ULONG      ulStructLength;          /* Structure length. */
    ULONG      ulClassNumber;          /* Class number. */
    CHAR       szmClassName[MIDI_NAME_LENGTH]; /* Class name. */
    ULONG      ulNumSlots;             /* Max. # of slots. */
    ULONG      ulAttributes;           /* Class attributes. */
} MIDICLASSINFO;

typedef MIDICLASSINFO *PMIDICLASSINFO;
```

MIDICLASSINFO Field - ulStructLength

ulStructLength ([ULONG](#))

Structure length. Must be set to the size of the MIDICLASSINFO structure before calling [MIDIQueryClassList](#).

MIDICLASSINFO Field - ulClassNumber

ulClassNumber ([ULONG](#))

Class number.

MIDICLASSINFO Field - szmClassName[MIDI_NAME_LENGTH]

szmClassName[MIDI_NAME_LENGTH] ([CHAR](#))

Class name.

MIDICLASSINFO Field - ulNumSlots

ulNumSlots ([ULONG](#))

Maximum number of slots instances of this class support.

MIDICLASSINFO Field - ulAttributes

ulAttributes ([ULONG](#))

Class attributes. Currently this will be zero.

MIDIHEADER

This structure is a MIDI subheader and is usually contained within the [MMMIDIHEADER](#) structure. It is part of the standard presentation format for MIDI data and is returned on an [mmioGetHeader](#) function and is used on the [mmioSetHeader](#) function when the [HMMIO](#) refers to an open MIDI file.

```
typedef struct _MIDIHEADER {
    CHAR        chHeaderChunk[4];      /* Chunk type. */
    ULONG       ulHeaderLength;        /* Header length. */
    USHORT      usFormat;              /* File format. */
    USHORT      usNumTracks;           /* Number of tracks. */
    USHORT      usDivision;            /* Time format. */
    VOID        *vpAdditionalInformation; /* Additional info. */
} MIDIHEADER;

typedef MIDIHEADER *PMIDIHEADER;
```

MIDIHEADER Field - chHeaderChunk[4]

chHeaderChunk[4] ([CHAR](#))

The characters used to identify the chunk type. They will always be M, C, H, and D.

MIDIHEADER Field - ulHeaderLength

ulHeaderLength ([ULONG](#))

Length of the chunk as previously described. Under most scenarios the value would be 6.

MIDIHEADER Field - usFormat

usFormat (USHORT)

The format of the file. The currently accepted values are 0 and 1. All 0 format files have one track of information. Format 1 files can have more than one track of information.

MIDIHEADER Field - usNumTracks

usNumTracks (USHORT)

The number of tracks in the file. For format 0 this would be 1 track.

MIDIHEADER Field - usDivision

usDivision (USHORT)

The time format of the file. If the high-order bit is 0, the time format is ticks per Quarter note. Otherwise the time format is SMPTE, with the lower 8 bits representing ticks per frame and bits 8-14 representing the negative SMPTE format. If you need more information on the time format of the file, refer to *The Standard MIDI Files* published by the International MIDI Association.

MIDIHEADER Field - *vpAdditionalInformation

*vpAdditionalInformation

A pointer to additional MIDI information.

MIDIINSTANCEINFO

This structure contains fields for the [MIDIQueryInstanceList](#) function.

```
typedef struct _MIDIINSTANCEINFO {
    ULONG      ulStructLength;      /* Structure length. */
    MINSTANCE  minstance;           /* Instance number. */
    ULONG      ulClassNumber;       /* Class number. */
    CHAR       szmInstanceName[MIDI_NAME_LENGTH]; /* Instance name. */
    ULONG      ulNumLinks;          /* Number of links. */
    ULONG      ulAttributes;        /* Instance attributes. */
} MIDIINSTANCEINFO;

typedef MIDIINSTANCEINFO *PMIDIINSTANCEINFO;
```

MIDIINSTANCEINFO Field - ulStructLength

ulStructLength ([ULONG](#))
Structure length. Must be set to the size of the MIDIINSTANCEINFO structure before calling [MIDIQueryInstanceList](#).

MIDIINSTANCEINFO Field - minstance

minstance ([MINSTANCE](#))
Instance number.

MIDIINSTANCEINFO Field - ulClassNumber

ulClassNumber ([ULONG](#))
Class number.

MIDIINSTANCEINFO Field - szmInstanceName[MIDI_NAME_LENGTH]

szmInstanceName[MIDI_NAME_LENGTH] ([CHAR](#))
Instance name.

MIDIINSTANCEINFO Field - ulNumLinks

ulNumLinks ([ULONG](#))
Number of links.

MIDIINSTANCEINFO Field - ulAttributes

ulAttributes ([ULONG](#))
Instance attributes.

MIDI_INST_ATTR_CAN_RECV
Instance can accept messages.

MIDI_INST_ATTR_CAN_SEND

Instance can send messages.
MIDI_INST_ATTR_ENABLE_R
Receive is enabled.
MIDI_INST_ATTR_ENABLE_S
Send is enabled.

MIDISETUP

This structure contains fields for the [MIDISetup](#) function.

```
typedef struct _MIDISETUP {  
    ULONG          ulStructLength;          /* Structure length. */  
    PULONG         pulMaxRTSysexLength;     /* Max. size of message. */  
    PPULONG        ppulMIDICurrentTime;     /* Current time. */  
    ULONG          ulFlags;                 /* Indicates callback type. */  
    PFNMIDI_NOTIFYCALLBACK pfnMIDI_NotifyCallback; /* Notification callbacks. */  
    HWND           hwndCallback;            /* Window handle for callback. */  
    HQUEUE         hqueueCallback;          /* Queue handle for callback. */  
} MIDISETUP;  
  
typedef MIDISETUP *PMIDISETUP;
```

MIDISETUP Field - ulStructLength

ulStructLength ([ULONG](#))

Structure length. Must be set to the size of the MIDISETUP structure before calling [MIDISetup](#).

MIDISETUP Field - pulMaxRTSysexLength

pulMaxRTSysexLength ([PULONG](#))

Maximum size of a real-time system exclusive (SysEx) message. Although there is currently no support for SysEx messages, this field must point to valid memory because a value is returned.

MIDISETUP Field - ppulMIDICurrentTime

ppulMIDICurrentTime ([PPULONG](#))

Pointer to a pointer to the RTMIDI timer.

MIDISETUP Field - ulFlags

ulFlags ([ULONG](#))
This field is reserved and must be set to 0.

MIDISETUP Field - pfnMIDI_NotifyCallback

pfnMIDI_NotifyCallback ([PFNMIDI_NOTIFYCALLBACK](#))
Function for notification callbacks. This field is currently ignored.

MIDISETUP Field - hwndCallback

hwndCallback ([HWND](#))
Window handle for callback. This field is currently ignored.

MIDISETUP Field - hqueueCallback

hqueueCallback ([HQUEUE](#))
Queue handle for callback. This field is currently ignored.

MINSTANCE

A MIDI instance.

```
typedef ULONG MINSTANCE;
```

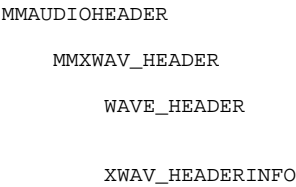
MMAUDIOHEADER

This structure is the audio header and it describes attributes of an open audio file. It is the standard presentation format for audio data and is returned on an [mmioGetHeader](#) function call and is used on an [mmioSetHeader](#) function when the [HMMIO](#) refers to an open audio file or to an open movie file when the active track for this file is set to an audio track.

```
typedef struct _MMAUDIOHEADER {
    ULONG          ulHeaderLength; /* Header length in bytes. */
    ULONG          ulContentType;  /* Audio content type. */
    ULONG          ulMediaType;    /* Media type. */
    MMXWAV_HEADER  mmXWAVHeader;   /* Header. */
}
```

```
} MMAUDIOHEADER;  
  
typedef MMAUDIOHEADER *PMMAUDIOHEADER;
```

Structure Layout for MMAUDIOHEADER



MMAUDIOHEADER Field - ulHeaderLength

ulHeaderLength (ULONG)
Length of this header, in bytes.

MMAUDIOHEADER Field - ulContentType

ulContentType (ULONG)
Audio content type. Acceptable values for this field are as follows:

- MMIO_MIDI_UNKNOWN
Unknown MIDI content.
- MMIO_MIDI_VOICE
Limited range.
- MMIO_MIDI_MUSIC
FM radio or equivalent.
- MMIO_MIDI_HIFI
High quality recording.

MMAUDIOHEADER Field - ulMediaType

ulMediaType (ULONG)
The type of multimedia data in the file. This will be determined by MMIO and returned to the application. The currently defined value is:

MMIO_MEDIATYPE_AUDIO

MMAUDIOHEADER Field - mmXWAVHeader

mmXWAVHeader ([MMXWAV_HEADER](#))

Contains specific information about the digital audio content including the format of the data representation for the data that it is associated.

MMCFINFO

This structure contains information about a RIFF compound file. This structure is returned on the [mmioCFGetInfo](#) function and used on the [mmioCFSetInfo](#) function.

```
typedef struct _MMCFINFO {
    ULONG      ulHeaderSize;      /* CTOC header size. */
    ULONG      ulEntriesTotal;    /* Number of CTOC table entries. */
    ULONG      ulEntriesDeleted;  /* Number of CTOC table entries deleted. */
    ULONG      ulEntriesUnused;   /* Number of unused CTOC entries. */
    ULONG      ulBytesTotal;      /* Combined byte size of all CGRP elements. */
    ULONG      ulBytesDeleted;    /* Byte size of all deleted elements in CGRP. */
    ULONG      ulHeaderFlags;     /* Flags. */
    USHORT     usEntrySize;       /* Size of each CTOC table entry. */
    USHORT     usNameSize;        /* Size of name field in entry (default 13). */
    USHORT     usExHdrFields;     /* Number of CTOC header extra fields. */
    USHORT     usExEntFields;     /* Number of CTOC entry extra fields. */
} MMCFINFO;

typedef MMCFINFO *PMMCFINFO;
```

Note: There are three optional fields that may follow this structure. They are variable-length arrays of information.

```
ULONG      (*aulExHdrFldUsage)[ ]; /* Array of header extra usage fields */
ULONG      (*aulExtEntFldUsage)[ ]; /* Array of entry extra usage fields */
ULONG      (*aulExHdrField)[ ]; /* Array of extra header fields */
```

Related Datatypes

- [MMIOINFO](#)
- [MMCTOCENTRY](#)

MMCFINFO Field - ulHeaderSize

ulHeaderSize ([ULONG](#))

The size of the compound table of contents (CTOC) header, including the variable-length arrays *aulExHdrFldUsage*, *aulExtEntFldUsage*, and *aulExHdrField*.

MMCFINFO Field - ulEntriesTotal

ulEntriesTotal ([ULONG](#))
The total number of CTOC table entries, including unused entries and entries corresponding to deleted compound-file elements.

MMCFINFO Field - ulEntriesDeleted

ulEntriesDeleted ([ULONG](#))
The number of CTOC table entries that refer to deleted CGRP elements.

MMCFINFO Field - ulEntriesUnused

ulEntriesUnused ([ULONG](#))
Number of unused CTOC entries.

MMCFINFO Field - ulBytesTotal

ulBytesTotal ([ULONG](#))
The combined size of all elements in the CGRP, including deleted elements.

MMCFINFO Field - ulBytesDeleted

ulBytesDeleted ([ULONG](#))
The combined size of all deleted elements in the CGRP.

MMCFINFO Field - ulHeaderFlags

ulHeaderFlags ([ULONG](#))
Flags that give information about the entire file.

CTOC_HF_SEQUENTIAL

Indicates that the valid entries in the CTOC table (that is, the entries that are neither deleted nor unused) are in the same order as the corresponding elements in the CGRP. If this bit is not set, the CTOC table entries might be in an arbitrary order.

CTOC_HF_MEDSUBTYPE

If this flag is set (bit value 1), then the *ulMedUsage* field of each CTOC table entry contains a four-character code that is a media element subtype. If the flag is not set (bit value 0), the *ulMedUsage* field is simply 32 bits of information that is interpreted as defined by the form type.

MMCFINFO Field - usEntrySize

usEntrySize (USHORT)

The size of each CTOC table entry. Each entry is the same size.

MMCFINFO Field - usNameSize

usNameSize (USHORT)

The size of the *pszElementName* field of each CTOC table entry, including the terminating NULL. Each *pszElementName* field must be padded with NULL to this length, and there must be at least one NULL.

MMCFINFO Field - usExHdrFields

usExHdrFields (USHORT)

The number of extra header fields; that is, the number of items in the arrays *aulExHdrFldUsage* and *aulExHdrField*.

MMCFINFO Field - usExEntFields

usExEntFields (USHORT)

The number of extra entry fields; that is, the number of items in the arrays *aulExEntFldUsage* and *aulExEntField*, of each CTOC table entry.

MMCKINFO

This structure contains information about a chunk in a RIFF file. It is used on the [mmioCreateChunk](#), [mmioDescend](#), and [mmioAscend](#) functions calls.


```
typedef struct _MMCKINFO {
    FOURCC    ckid;          /* Chunk ID (FOURCC). */
    ULONG     ckSize;        /* Chunk size (bytes). */
    FOURCC     fccType;      /* FOURCC type (if ckid RIFF or LIST). */
    ULONG     ulDataOffset;  /* File offset of data portion of chunk. */
    ULONG     ulFlags;       /* MMIO_DIRTY (if new chunk). */
} MMCKINFO;

typedef MMCKINFO *PMMCKINFO;
```

Related Datatypes

- [MMIOINFO](#)

MMCKINFO Field - ckid

ckid ([FOURCC](#))
Chunk ID (FOURCC).

MMCKINFO Field - ckSize

ckSize ([ULONG](#))
The size of the data portion of the chunk. It does not include the four-byte chunk ID, the four-byte chunk size, or the pad byte (if present).

MMCKINFO Field - fccType

fccType ([FOURCC](#))
The form type (for RIFF chunks) or list type (for LIST types).

MMCKINFO Field - ulDataOffset

ulDataOffset ([ULONG](#))
The file offset of the beginning of the data portion on the chunk (8 bytes from the beginning of the chunk header).

MMCKINFO Field - ulFlags

ulFlags ([ULONG](#))

MMIO_DIRTY indicates the chunk was created by [mmioCreateChunk](#)

MMCOMPRESS

This structure contains information used on [MMIOM_COMPRESS](#) and [MMIOM_CODEC_COMPRESS](#) messages.

```
typedef struct _MMCOMPRESS {
    ULONG    ulStructLen;    /* Length of this structure. */
    ULONG    ulFlags;        /* Command and status flags. */
    ULONG    ulSrcBufLen;    /* Source buffer size. */
    PVOID    pSrcBuf;        /* Source buffer. */
    ULONG    ulDstBufLen;    /* Destination buffer length. */
    PVOID    pDstBuf;        /* Destination buffer. */
    PVOID    pRunTimeInfo;   /* Control information. */
} MMCOMPRESS;

typedef MMCOMPRESS *PMMCOMPRESS;
```

MMCOMPRESS Field - ulStructLen

ulStructLen ([ULONG](#))

Indicates the length of entire structure.

MMCOMPRESS Field - ulFlags

ulFlags ([ULONG](#))

Some flags are set to instruct the I/O or CODEC procedure to act. Some flags are status returned from the I/O or CODEC procedure.

MMIO_IS_KEY_FRAME

This bit is set by the application to instruct the I/O procedure to compress the *pSrcBuf* into a key or reference frame. If the bit is not set, a delta frame is compressed.

MMIO_IS_PALETTE

A video palette is provided. Set by the application.

MMCOMPRESS Field - ulSrcBufLen

ulSrcBufLen ([ULONG](#))

The length in bytes of the buffer pointed to by the *pSrcBuf* field. On return, the length is updated to reflect the remaining number of compressed data bytes that have been processed.

MMCOMPRESS Field - pSrcBuf

pSrcBuf ([PVOID](#))

Pointer to the source uncompressed data.

MMCOMPRESS Field - ulDstBufLen

ulDstBufLen ([ULONG](#))

The length in bytes of the buffer pointed to by the *pDstBuf* field. On return, the length is updated to reflect the compressed data pointed to by *pDstBuf*.

MMCOMPRESS Field - pDstBuf

pDstBuf ([PVOID](#))

Pointer to the destination buffer that will contain compressed data.

MMCOMPRESS Field - pRunTimeInfo

pRunTimeInfo ([PVOID](#))

Control information pointer to [MMVIDEOCOMPRESS](#).

MMCTOCENTRY

This structure contains information about an entry in the compound table of contents (CTOC) of a RIFF compound file.

```
typedef struct _MMCTOCENTRY {
    ULONG    ulOffset;           /* Offset of element within CGRP. */
    ULONG    ulSize;             /* Size of element. */
    ULONG    ulMedType;          /* FOURCC of element. */
    ULONG    ulMedUsage;         /* Possible subtype. */
    ULONG    ulCompressTech;     /* Compression technique used. */
    ULONG    ulUncompressBytes; /* Actual size of uncompressed element. */
} MMCTOCENTRY;

typedef MMCTOCENTRY *PMMCTOCENTRY;
```

Note: There are two optional fields, a variable-length name field and ULONG array, that may follow this structure.

```
ULONG      (*aulExEntField)[ ];    /* Array of extra entry fields */
PSZ        pszElementName[ ];     /* Name of variable-length element */
```

Related Data Type

- [MMCFINFO](#)

MMCTOCENTRY Field - ulOffset

ulOffset ([ULONG](#))

The offset of the compound-file element within the compound-file resource-group (CGRP) chunk. This offset is measured from the beginning of the data portion of the chunk. For example, if *ulOffset* is 1000, and the chunk ID of the CGRP chunk is at offset 500, then the element is at offset 1508 (the 8 is because the chunk ID and chunk size occupy 4 bytes each).

MMCTOCENTRY Field - ulSize

ulSize ([ULONG](#))

The size that the element occupies in the CGRP chunk.

MMCTOCENTRY Field - ulMedType

ulMedType ([ULONG](#))

The four-character code of the media element type of the compound-file entry that the CTOC entry refers to. This field can be 0 if the compound-file entry is not to be interpreted as a stand-alone file. If the compound-file entry is a RIFF form, then the media element type is the same as the RIFF form type.

MMCTOCENTRY Field - ulMedUsage

ulMedUsage ([ULONG](#))

If the CTOC_HF_MEDSUBTYPE flag is present in the *ulHeaderFlags* field of the CTOC header ([MMCFINFO](#)), then the *ulMedUsage* field of each CTOC table entry (MMCTOCENTRY) contains a four-character code that is a media element subtype. If the flag is not set, the *ulMedUsage* field is simply 32 bits of information, which is defined by the form type.

MMCTOCENTRY Field - ulCompressTech

ulCompressTech (ULONG)

The compression technique used to compress the compound-file element. If this value is 0, the element is not compressed.

MMCTOCENTRY Field - ulUncompressBytes

ulUncompressBytes (ULONG)

The number of bytes that the compound-file element will occupy in memory after it has been decompressed by the decompression code associated with compression technique, *ulCompressTech* field of MMCTOCENTRY. If the *ulCompressTech* field is 0, the compound-file element is not compressed, and the *ulUncompressBytes* field of MMCTOCENTRY equals the *ulSize* field of MMCTOCENTRY, the file size of the compound-file element.

MMDECOMPRESS

This structure contains information used on an [MMIOM_DECOMPRESS](#) and [MMIOM_CODEC_DECOMPRESS](#) message call to a CODEC.

```
typedef struct _MMDECOMPRESS {
    ULONG    ulStructLen;    /* Structure length. */
    ULONG    ulFlags;        /* Command and status flags. */
    ULONG    ulSrcBufLen;    /* Source buffer size. */
    PVOID    pSrcBuf;        /* Source buffer. */
    ULONG    ulDstBufLen;    /* Destination buffer length. */
    PVOID    pDstBuf;        /* Destination buffer. */
    PVOID    pRunTimeInfo;   /* Control information. */
} MMDECOMPRESS;

typedef MMDECOMPRESS *PMMDECOMPRESS;
```

MMDECOMPRESS Field - ulStructLen

ulStructLen (ULONG)

Indicates the length of entire structure.

MMDECOMPRESS Field - ulFlags

ulFlags (ULONG)

Some flags are set to instruct the I/O or CODEC procedure to act. Others are status, returned from the I/O or CODEC procedure.

MMIO_DROP_DELTA_FRAME

Tells the I/O or CODEC procedure to drop the delta frame if the *pSrcBuf* field contains a delta frame. On return, the bit is reset if the delta frame is dropped.

MMIO_IS_KEY_FRAME

This bit is set by the I/O or CODEC procedure when the data contained in the *pSrcBuf* field is a key or reference frame.

MMIO_IS_PALETTE

A video palette is found. Set by the I/O procedure.

MMIO_PALETTE_CHANGE

The physical palette has been changed in the *genpalPhysical* field of the [MMVIDEODECOMPRESS](#) structure. Set by the application.

MMIO_ORIGIN_LOWERLEFT

Tells the I/O or CODEC procedure to decompress data to the destination buffer using the lower left as window origin. This means the decompressed data lines will be stored in reverse order in the destination buffer.

MMIO_ORIGIN_UPPERLEFT

Tells the I/O or CODEC procedure to decompress data to the destination buffer using the upper-left as window origin. This means the decompressed data lines will be stored in the same order as the incoming compressed data lines.

MMIO_RECTL_CHANGE

The movie rectangle has been changed. Valid rectangles are specified in the *prectl* and *ulRectlCount* fields of the [MMVIDEODECOMPRESS](#) structure.

MMDECOMPRESS Field - ulSrcBufLen

ulSrcBufLen ([ULONG](#))

The length in bytes of the buffer pointed to by the *pSrcBuf* field. On return, the length is updated to reflect the remaining number of compressed data bytes that need to be processed. Zero must be returned when all bytes are decompressed.

MMDECOMPRESS Field - pSrcBuf

pSrcBuf ([PVOID](#))

Pointer to the source compressed data. On return the pointer is adjusted to point to the data not yet compressed. The NULL must be returned when all data in the *pSrcBuf* field is decompressed.

MMDECOMPRESS Field - ulDstBufLen

ulDstBufLen ([ULONG](#))

The length in bytes of the buffer pointed to by *pDstBuf*. On return, the length is updated to reflect the compressed data pointed to by the *pDstBuf* field.

MMDECOMPRESS Field - pDstBuf

pDstBuf ([PVOID](#))

Pointer to the destination uncompressed data.

MMDECOMPRESS Field - pRunTimeInfo

pRunTimeInfo ([PVOID](#))

Pointer to [MMVIDEODECOMPRESS](#). Contains the CODEC runtime information.

MMDRV_OPEN_PARMS

This structure is passed from MDM to the MCD on the [MCI_OPEN](#) message. The *ulParam2* field of this structure contains the *pParam2* passed to MDM by the application.

```
typedef struct _MMDRV_OPEN_PARMS {
    HWND      hwndCallback;      /* Window handle for callback. */
    USHORT    usDeviceID;        /* The device ID assigned to this instance. */
    USHORT    usDeviceType;      /* Device type number. */
    USHORT    usDeviceOrd;       /* Device ordinal number. */
    PVOID     pInstance;         /* Pointer to the instance structure allocated. */
    CHAR      szDevDLLName[260]; /* Character string. */
    PSZ       pszElementName;    /* Typically a file name or NULL. */
    USHORT    usDevParamLen;     /* Device parameters data block length. */
    PVOID     pDevParam;         /* Device parameters data block. */
    USHORT    usResourceUnitsRequired; /* Number of resource units. */
    USHORT    usResourceClass;   /* Resource class. */
    USHORT    usResourcePriority; /* Resource priority. */
    ULONG     ulParam2;          /* Pointer to MCI_OPEN structure. */
} MMDRV_OPEN_PARMS;

typedef MMDRV_OPEN_PARMS *PMMDRV_OPEN_PARMS;
```

MMDRV_OPEN_PARMS Field - hwndCallback

hwndCallback ([HWND](#))

Window handle for callback.

MMDRV_OPEN_PARMS Field - usDeviceID

usDeviceID ([USHORT](#))

The device ID assigned to this instance.

MMDRV_OPEN_PARMS Field - usDeviceType

usDeviceType (USHORT)
Device type number.

MMDRV_OPEN_PARMS Field - usDeviceOrd

usDeviceOrd (USHORT)
Device ordinal number.

MMDRV_OPEN_PARMS Field - pInstance

pInstance (PVOID)
Pointer to the instance structure allocated.

MMDRV_OPEN_PARMS Field - szDevDLLName[260]

szDevDLLName[260] (CHAR)
Character string containing the device specific DLL name to call for the open (for example, ACPA.DLL).

MMDRV_OPEN_PARMS Field - pszElementName

pszElementName (PSZ)
Typically a file name or NULL.

MMDRV_OPEN_PARMS Field - usDevParmLen

usDevParmLen (USHORT)
Device parameters data block length.

MMDRV_OPEN_PARMS Field - pDevParm

pDevParm ([PVOID](#))

This data block is unique to each type of device (for example, LVD "COM1 9600 N 7 1").

MMDRV_OPEN_PARMS Field - usResourceUnitsRequired

usResourceUnitsRequired ([USHORT](#))

Number of resource units required for this instance.

MMDRV_OPEN_PARMS Field - usResourceClass

usResourceClass ([USHORT](#))

Resource class this instance belongs to.

MMDRV_OPEN_PARMS Field - usResourcePriority

usResourcePriority ([USHORT](#))

Resource priority for this instance.

MMDRV_OPEN_PARMS Field - ulParam2

ulParam2 ([ULONG](#))

Pointer to the MCI_OPEN data structure passed in by the application.

MMEXTENDINFO

This structure contains information used on an [mmioSet](#) function call to digital video I/O procedure. It contains information used to associate a CODEC with an open file or to query information about CODECs already loaded. This structure also provides a way of setting attributes used in reading and writing from a digital video movie file.

```
typedef struct _MMEXTENDINFO {
    ULONG    ulStructLen; /* Structure length. */
    ULONG    ulBufSize;  /* Total buffer size. */
    ULONG    ulFlags;     /* Flags. */
    ULONG    ulTrackID;   /* Track ID. */
    ULONG    ulNumCODECs; /* Number of CODEC entries. */
}
```

```
    PCODECASSOC    pCODECAssoc; /* Pointer to CODECASSOC. */
} MMEXTENDINFO;

typedef MMEXTENDINFO *PMMEXTENDINFO;
```

MMEXTENDINFO Field - ulStructLen

ulStructLen (ULONG)
Indicates the length of entire structure.

MMEXTENDINFO Field - ulBufSize

ulBufSize (ULONG)
Indicates the length of buffer in bytes to hold all of the information when the MMIO_QUERY_EXTENDINFO_ALL flag is set.

MMEXTENDINFO Field - ulFlags

ulFlags (ULONG)
Specifies the following flags:

MMIO_TRACK
If set, all MMIO operations will be performed on the active track specified in the *ulTrackID* field.

MMIO_RESETTRACKS
If set, turns off the active track number; track 0 is used as the current track.

MMIO_REVERSE_READ
Reserved for future use.

MMIO_SCAN_READ
If set, all MMIO operations will be performed in scan mode, meaning only key video frames are processed.

MMIO_NORMAL_READ
If set, all MMIO operations will be performed in Normal mode, which processes all frames.

MMIO_CODEC_ASSOC
If set, the *pCODECAssoc* and *ulNumCODECs* fields are used for setting the active CODECs or returning the CODEC information.

MMEXTENDINFO Field - ulTrackID

ulTrackID (ULONG)
The active track identifier set to the file. If MMIO_RESETTRACKS is specified, all MMIO operations are performed on the file basis.

MMEXTENDINFO Field - ulNumCODECs

ulNumCODECs ([ULONG](#))

Specifies the number of entries in the buffer pointed to by the [CODECASSOC](#) structure.

MMEXTENDINFO Field - pCODECAssoc

pCODECAssoc ([PCODECASSOC](#))

Pointer to the [CODECASSOC](#) structure.

MMFORMATINFO

This structure describes some of the characteristics of an I/O procedure and the type of data that it can process. It is used with the [mmioGetFormatName](#), [mmioGetFormats](#), and [mmioQueryFormatCount](#) functions.

```
typedef struct _MMFORMATINFO {
    ULONG      ulStructLen;           /* Length of this structure. */
    FOURCC     fccIOProc;            /* IOProc identifier. */
    ULONG      ulIOProcType;         /* Type of IOProc. */
    ULONG      ulMediaType;          /* Media type. */
    ULONG      ulFlags;              /* IOProc capability flags. */
    CHAR       szDefaultFormatExt[sizeof(FOURCC)+1]; /* Default extension 4 + null. */
    ULONG      ulCodePage;           /* Code page. */
    ULONG      ulLanguage;           /* Language. */
    LONG       lNameLength;          /* Length of identifier string. */
} MMFORMATINFO;

typedef MMFORMATINFO *PMMFORMATINFO;
```

MMFORMATINFO Field - ulStructLen

ulStructLen ([ULONG](#))

Indicates the length of the entire structure, allowing applications to detect changes in future versions.

MMFORMATINFO Field - fccIOProc

fccIOProc (FOURCC)

The four-character code that uniquely identifies the I/O procedure that is responsible for processing this file format.

MMFORMATINFO Field - ulIOProcType

ulIOProcType (ULONG)

Specifies the type of I/O procedure. These include:

MMIO_IOPROC_STORAGESYSTEM
MMIO_IOPROC_FILEFORMAT
MMIO_IOPROC_DATAFORMAT

MMFORMATINFO Field - ulMediaType

ulMediaType (ULONG)

The type of multimedia data in the file. This will be determined by MMIO and returned to the application. The currently defined values are:

MMIO_MEDIATYPE_AUDIO
Audio media

MMIO_MEDIATYPE_IMAGE
Image media

MMIO_MEDIATYPE_DIGITALVIDEO
Digital video media

MMIO_MEDIATYPE_MIDI
MIDI media

MMIO_MEDIATYPE_MOVIE
Contains digital video and audio data.

MMIO_MEDIATYPE_ANIMATION
Currently not supported.

MMIO_MEDIATYPE_COMPOUND
Compound media

MMIO_MEDIATYPE_OTHER
Known media type, but not currently defined by the Toolkit.

MMIO_MEDIATYPE_UNKNOWN
Unknown media

MMFORMATINFO Field - ulFlags

ulFlags (ULONG)

These flags represent capabilities of the I/O procedure. These flags are set only by the I/O procedure. Each flag represents one bit. A value of 1 indicates that capability is supported. The calling routine should check these flags to ensure the I/O procedure can accommodate the request.

MMIO_CANREADTRANSLATED
MMIO_CANREADUNTRANSLATED
MMIO_CANREADWRITETRANSLATED
MMIO_CANREADWRITEUNTRANSLATED
MMIO_CANWRITETRANSLATED
MMIO_CANWRITEUNTRANSLATED
MMIO_CANSEEKTRANSLATED
MMIO_CANSEEKUNTRANSLATED
MMIO_CANINSERTTRANSLATED
MMIO_CANINSERTUNTRANSLATED
MMIO_CANSAVETRANSLATED
MMIO_CANSAVEUNTRANSLATED
MMIO_CANMULTITRACKREADTRANSLATED
MMIO_CANMULTITRACKREADUNTRANSLATED
MMIO_CANMULTITRACKWRITETRANSLATED
MMIO_CANMULTITRACKWRITEUNTRANSLATED
MMIO_CANTRACKSEEKTRANSLATED
MMIO_CANTRACKSEEKUNTRANSLATED
MMIO_CANTRACKREADTRANSLATED
MMIO_CANTRACKREADUNTRANSLATED
MMIO_CANTRACKWRITETRANSLATED
MMIO_CANTRACKWRITEUNTRANSLATED

MMFORMATINFO Field - szDefaultFormatExt[sizeof(FOURCC)+1]

szDefaultFormatExt[sizeof(FOURCC)+1] ([CHAR](#))

A 4-character, null-terminated string containing the file name extension normally associated with this file format. This allows applications creating media files to use the most appropriate naming conventions.

MMFORMATINFO Field - ulCodePage

ulCodePage ([ULONG](#))

The codepage and country code of the format name string for this I/O procedure.

As described for RIFF compound files in the IBM-Microsoft Joint Specification, the low-order word contains either:

- Zero (0): Use standard codepage.
- Codepage: Specific codepage to be used.

The high-order word contains either:

- Zero (0): Ignore this field.
- Country code: Specific country code.

MMFORMATINFO Field - ulLanguage

ulLanguage (ULONG)

Provides the language and dialect codes of the format name string for this I/O procedure.

As described for RIFF compound-files in the IBM-Microsoft Joint Specification, the low-order word contains either:

- Zero (0): Ignore this field
- Language: Specific language to be used

The high-order word contains either:

- Zero (0): Ignore this field
- Reserved:

MMFORMATINFO Field - INameLength

INameLength (LONG)

Length of format identifier string, *not* including the null-terminating character.

MMIMAGEHEADER

This structure is the image header and it describes attributes of an open image file. It is the standard presentation format for image data and is returned on an [mmioGetHeader](#) function and is used on the [mmioSetHeader](#) function when the [HMMIO](#) refers to an open image file.

Note: Valid field values for the standard image presentation format are defined where it is appropriate.

```
typedef struct _MMIMAGEHEADER {
    ULONG          ulHeaderLength;           /* Length in bytes. */
    ULONG          ulContentType;           /* Image content. */
    ULONG          ulMediaType;             /* Media type. */
    MMXDIBHEADER   mmXDIBHeader;           /* PM compatible header. */
    RGB2           bmiColors[MAX_PALETTE]; /* PM compatible palette. */
} MMIMAGEHEADER;

typedef MMIMAGEHEADER *PMMIMAGEHEADER;
```

Structure Layout for MMIMAGEHEADER

```
MMIMAGEHEADER
    MMXDIBHEADER
        XDIBHDR_PREFIX
            BITMAPINFOHEADER2
```

MMIMAGEHEADER Field - ulHeaderLength

ulHeaderLength (ULONG)
The length of this header in bytes.

MMIMAGEHEADER Field - ulContentType

ulContentType (ULONG)
The image content type. The acceptable values for this field are as follows:

- MMIO_IMAGE_UNKNOWN
Unknown image content.
- MMIO_IMAGE_DRAWING
Simple drawing.
- MMIO_IMAGE_GRAPH
Graphs and cartoons.
- MMIO_IMAGE_PHOTO
Varying color and shades.

MMIMAGEHEADER Field - ulMediaType

ulMediaType (ULONG)
The type of multimedia data in the file. This will be determined by MMIOand returned to the application. The currently defined value is:

- MMIO_MEDIATYPE_IMAGE

MMIMAGEHEADER Field - mmXDIBHeader

mmXDIBHeader ([MMXDIBHEADER](#))

Extension of OS/2 2.0 PM compatible header.

MMIMAGEHEADER Field - bmiColors[MAX_PALETTE]

bmiColors[MAX_PALETTE] ([RGB2](#))

PM compatible palette (256 entries).

MMINIFILEINFO

The structure contains information about an I/O procedure entry in the MMPMMMIO.INI file. Each entry describes an I/O procedure that can be used by the system. Each I/O procedure has one entry in the MMPMMMIO.INI file. This structure is used on the [mmioIniFileHandler](#) function.

```
typedef struct _MMINIFILEINFO {
    FOURCC      fccIOProc;          /* Identifies IOProc. */
    CHAR        szDLLName[DLLNAME_SIZE]; /* DLL name of IOProc. */
    CHAR        szProcName[PROCNAME_SIZE]; /* Entry point of DLL. */
    ULONG       ulFlags;             /* Reserved. */
    ULONG       ulExtendLen;         /* Length of extended fields. */
    ULONG       ulMediaType;         /* Media type. */
    ULONG       ulIOProcType;        /* IOProc type. */
    CHAR        szDefExt[sizeof(MAX_EXTENSION_NAME)]; /* Default file extension. */
} MMINIFILEINFO;

typedef MMINIFILEINFO *PMMINIFILEINFO;
```

MMINIFILEINFO Field - fccIOProc

fccIOProc ([FOURCC](#))

The four-character code that identifies the file's I/O procedure.

MMINIFILEINFO Field - szDLLName[DLLNAME_SIZE]

szDLLName[DLLNAME_SIZE] ([CHAR](#))

The DLL name of the I/O procedure.

MMINIFILEINFO Field - szProcName[PROCNAME_SIZE]

szProcName[PROCNAME_SIZE] (CHAR)
The procedure entry point of the DLL.

MMINIFILEINFO Field - ulFlags

ulFlags (ULONG)
Reserved

MMINIFILEINFO Field - ulExtendLen

ulExtendLen (ULONG)
Length of extended fields.

MMINIFILEINFO Field - ulMediaType

ulMediaType (ULONG)
Media type.

MMINIFILEINFO Field - ullIOProcType

ullIOProcType (ULONG)
The type of I/O procedure.

MMINIFILEINFO Field - szDefExt[sizeof(MAX_EXTENSION_NAME)]

szDefExt[sizeof(MAX_EXTENSION_NAME)] (CHAR)
Default file extension.

MMIO_EDIT_PARMS

This structure contains information for performing edit operations and is used with the [MMIOM_CLEAR](#), [MMIOM_COPY](#), [MMIOM_CUT](#), and [MMIOM_PASTE](#) messages.

```
typedef struct _MMIO_EDIT_PARMS {  
    ULONG      ulStrucLen;          /* Length of this structure. */  
    HWND       hwndWindow;         /* Window handle. */  
    ULONG      ulStartTime;        /* Starting time in microseconds. */  
    ULONG      ulDuration;         /* Duration in microseconds. */  
    ULONG      ulCurrentFilePosition; /* Current file position in microseconds. */  
    ULONG      ulNewFilePosition;   /* New file position in microseconds. */  
    ULONG      ulNewFileLength;     /* New file length in microseconds. */  
    PVOID      pBuffer;            /* User buffer. */  
    PVOID      pBufferLength;       /* Length of user buffer. */  
    PVOID      pHeader;            /* Describes buffer. */  
    BOOL       fUseBuffer;         /* Use buffer instead of clipboard. */  
} MMIO_EDIT_PARMS;  
  
typedef MMIO_EDIT_PARMS *PMMIO_EDIT_PARMS;
```

MMIO_EDIT_PARMS Field - ulStrucLen

ulStrucLen (ULONG)
The total length of this structure in bytes. The length includes the four bytes for this field.

MMIO_EDIT_PARMS Field - hwndWindow

hwndWindow (HWND)
The window handle associated with the current editing operations.

This handle is used in clipboard-related PM calls, such as WinSetClipbrdOwner.

MMIO_EDIT_PARMS Field - ulStartTime

ulStartTime (ULONG)
The starting time for the operation, expressed in microseconds.

The I/O procedure uses this time to calculate the first element (movie frame, audio byte, and so on) to be included in the editing operation or the position for an insert.

MMIO_EDIT_PARMS Field - ulDuration

ulDuration (ULONG)

The length of time to be included in the editing operation.

The I/O procedure uses *ulStartTime* and *ulDuration* fields to calculate the last element to be included in the editing operation.

This parameter is inclusive. If the duration falls in any portion of time occupied by an element, the entire element will be included in the operation.

MMIO_EDIT_PARMS Field - ulCurrentFilePosition

ulCurrentFilePosition (ULONG)

The current position of the file in microseconds, as viewed by the MCD, at the time the editing operation is processed.

This field is used by the I/O procedure to calculate the starting element for the editing operation when the FROM parameter is omitted from the media control interface message.

MMIO_EDIT_PARMS Field - ulNewFilePosition

ulNewFilePosition (ULONG)

The new file position in microseconds. The I/O procedure returns the new file position to be set by the MCD, by way of a seek, so that the media is correctly positioned. The media control interface editing messages define the correct new media position.

MMIO_EDIT_PARMS Field - ulNewFileLength

ulNewFileLength (ULONG)

The new length of the file in microseconds after the editing operation is performed.

The MCD must update any private data structures, such as copies of the file header, to reflect this new media length.

MMIO_EDIT_PARMS Field - pBuffer

pBuffer (PVOID)

This field is set from the *pBuff* field of the [MCI_EDIT_PARMS](#) structure if the MCI_TO_BUFFER flag is specified; this field is NULL if the MCI_TO_BUFFER flag is not specified.

MMIO_EDIT_PARMS Field - pBufferLength

pBufferLength (PVOID)

This field is the length of *pBuffer* and is set from the *ulBufLen* field of the [MCI_EDIT_PARMS](#) structure.

MMIO_EDIT_PARMS Field - pHeader

pHeader (PVOID)

This field is set from the *pHeader* field from the [MCI_EDIT_PARMS](#) structure.

MMIO_EDIT_PARMS Field - fUseBuffer

fUseBuffer (BOOL)

Use buffer instead of clipboard.

MMIOINFO

This structure contains the current state information of an open file. It is returned on the [mmioGetInfo](#) function and used on the [mmioSetInfo](#) function.

```
typedef struct _MMIOINFO {
    ULONG        ulFlags;           /* Open flags. */
    FOURCC       fccIOProc;        /* FOURCC of the IOProc to use. */
    PMMIOPROC    pIOProc;          /* Function pointer to IOProc to use. */
    ULONG        ulErrorRet;       /* Extended error return code. */
    LONG         cchBuffer;        /* I/O buffer size (if used). */
    PCHAR        pchBuffer;       /* Start of I/O buffer. */
    PCHAR        pchNext;         /* Next byte to read or write in buffer. */
    PCHAR        pchEndRead;       /* Last byte in buffer can be read. */
    PCHAR        pchEndWrite;     /* Last byte in buffer can be written. */
    LONG         lBufOffset;       /* Offset in buffer to pchNext. */
    LONG         lDiskOffset;      /* Disk offset in file. */
    ULONG        aulInfo[4];       /* IOProc specific field. */
    LONG         lLogicalFilePos;  /* Actual file position, buffered or not. */
    ULONG        ulTranslate;      /* Translation field. */
    FOURCC       fccChildIOProc;   /* FOURCC of child IOProc. */
    PVOID        pExtraInfoStruct; /* Pointer to related structure. */
    HMMIO        hmmio;           /* Handle to media element. */
} MMIOINFO;

typedef MMIOINFO *PMMIOINFO;
```

MMIOINFO Field - ulFlags

ulFlags (ULONG)

Access Mode

The access mode of the file, equal to one of the following values:

- MMIO_READ
The file was opened only for reading.
- MMIO_WRITE
The file was opened only for writing.
- MMIO_READWRITE
The file was opened for both reading and writing.

File Sharing Mode

The file sharing mode of the file, equal to one of the following values:

- MMIO_COMPAT
The file was opened with compatibility mode, allowing any process on a given system to open the file any number of times.
- MMIO_EXCLUSIVE
The file was opened with exclusive mode, denying other processes both read and write access to the file.
- MMIO_DENYWRITE
Other processes are denied write access to the file.
- MMIO_DENYREAD
Other processes are denied read access to the file.
- MMIO_DENYNONE
Other processes are not denied read or write access to the file.

Other Modes

Other modes of the file, equal to *one or more* of the following values:

- MMIO_CREATE
[mmioOpen](#) was directed to create the file, or truncate it to 0 length if it already existed.
- MMIO_ALLOCBUF
The file's I/O buffer was allocated by [mmioOpen](#) or [mmioSetBuffer](#).
- MMIO_BUFSHARED
Requests that if MMIO allocates the buffer, it does so from shared memory.
- MMIO_VERTBAR
Requests that the vertical bar symbol (|) be used as a file separator character rather than the plus sign (+).
- MMIO_DELETE
Directs [mmioOpen](#) to delete the file.
- MMIO_APPEND
Directs [mmioOpen](#) to allow appending to the end of the file.
- MMIO_DIRTY
Forces a write operation on an [mmioAdvance](#) function for low-level I/O.
- MMIO_NOIDENTIFY
Directs [mmioOpen](#) not to attempt an automatic identification on this file.

MMIOINFO Field - fccIOProc

fccIOProc ([FOURCC](#))
The four-character code that identifies the file's I/O procedure. If the I/O procedure is not an installed one, *fccIOProc* will be NULL. See [mmioOpen](#) for a list of the predefined I/O procedure identifiers.

MMIOINFO Field - pIOProc

pIOProc ([PMMIOPROC](#))
The address of the file's I/O procedure.

MMIOINFO Field - ulErrorRet

ulErrorRet ([ULONG](#))
Extended error return code.

MMIOINFO Field - cchBuffer

cchBuffer ([LONG](#))
The size of the file's I/O buffer (in bytes), or 0 if the file either does not have an I/O buffer or has a 0-byte buffer.

MMIOINFO Field - pchBuffer

pchBuffer ([PCHAR](#))
The address of the file's I/O buffer. If *pchBuffer* is NULL, the file is not buffered.

MMIOINFO Field - pchNext

pchNext ([PCHAR](#))
Points to the next byte in the I/O buffer to be read or written to. Points to *pchEndRead* if no more bytes can be read without calling [mmioAdvance](#) or [mmioRead](#). Points to *pchEndWrite* if no more bytes can be written without calling [mmioAdvance](#) or [mmioWrite](#).

MMIOINFO Field - pchEndRead

pchEndRead ([PCHAR](#))

The last byte in the buffer that can be read from.

MMIOINFO Field - pchEndWrite

pchEndWrite ([PCHAR](#))

The last byte in the buffer that can be written to.

MMIOINFO Field - IBufOffset

IBufOffset ([LONG](#))

Reserved for use by the MMIO functions internally.

MMIOINFO Field - IDiskOffset

IDiskOffset ([LONG](#))

The file offset that the next [mmioRead](#) or [mmioWrite](#) function, or [MMIOM_READ](#) or [MMIOM_WRITE](#) message (sent to the file's I/O procedure) will read or write to. Reserved for use by the MMIO functions internally.

MMIOINFO Field - aullInfo[4]

aullInfo[4] ([ULONG](#))

State information that is maintained by the I/O procedure. Certain I/O procedures also use these fields to transfer information from the caller to the I/O procedure when the file is opened. See [mmioOpen](#).

MMIOINFO Field - ILogicalFilePos

ILogicalFilePos ([LONG](#))

Actual logical-file position, whether buffered or not.

MMIOINFO Field - ulTranslate

ulTranslate (ULONG)

This field is the translation flag field and it has the following possible values:

MMIO_NOTTRANSLATE

Neither the Header or Data portion of the file is translated into the defined standard presentation format for the specified media type.

MMIO_TRANSLATEDATA

The Data portion of the file is translated into the defined standard presentation format for the specified media type.

MMIO_TRANSLATEHEADER

The Header portion of the file is translated into the defined standard presentation format for the specified media type.

MMIOINFO Field - fccChildIOProc

fccChildIOProc (FOURCC)

FOURCC structure of the child I/O procedure to use. In the current implementation, this is the storage system I/O procedure.

MMIOINFO Field - pExtraInfoStruct

pExtraInfoStruct (PVOID)

Pointer to a structure of related data to be used by the I/O procedure.

Set *pExtraInfoStruct* to point to [JPEGOPTIONS](#) for related JPEG I/O procedure information.

MMIOINFO Field - hmmio

hmmio (HMMIO)

The MMIO handle to the open file. I/O procedures can use this handle in calls to MMIO functions.

MMIO_STATUS_PARMS

This structure contains information used on an [MMIOM_STATUS](#) message.

```
typedef struct _MMIO_STATUS_PARMS {  
    HWND      hwndWindow; /* Window handle. */  
};
```



```
ULONG    ulReturn;    /* Return information. */
ULONG    ulItem;      /* Status item. */
ULONG    ulValue;     /* Status value. */
ULONG    ulType;      /* Format. */
} MMIO_STATUS_PARMS;
```

```
typedef MMIO_STATUS_PARMS *PMMIO_STATUS_PARMS;
```

MMIO_STATUS_PARMS Field - hWndWindow

hWndWindow ([HWND](#))

This field is required if a specific window handle is required to process a status request.

If this parameter is omitted and a window handle is needed, the I/O procedure will use `HWND_DESKTOP`.

MMIO_STATUS_PARMS Field - ulReturn

ulReturn ([ULONG](#))

Contains the status information upon return.

MMIO_STATUS_PARMS Field - ulItem

ulItem ([ULONG](#))

The status item to query.

MMIO_STATUS_PARMS Field - ulValue

ulValue ([ULONG](#))

The status value to extend the status structure.

MMIO_STATUS_PARMS Field - ulType

ulType ([ULONG](#))

An `MCI_FORMAT` flag is returned here, when required, to specify the format of the value returned in *ulReturn*; for example, the time format when a time value is returned.

MMIO_WINMSG

This structure contains information used on an [MMIOM_WINMSG](#) message.

Note: The fields of this structure correspond to the parameters of the WinSendMsg and WinPostMsg functions.

The digital video device sends WM_DESTROYCLIPBOARD messages to the I/O procedure. Since it ignores the return code from the message, movie file I/O procedures are not required to support [MMIOM_WINMSG](#).

```
typedef struct _MMIO_WINMSG {
    HWND      hwndWindow; /* Window handle. */
    USHORT    usMessage; /* PM message number. */
    PVOID     pParam1; /* First PM message parameter. */
    PVOID     pParam2; /* Second PM message parameter. */
} MMIO_WINMSG;

typedef MMIO_WINMSG *PMMIO_WINMSG;
```

MMIO_WINMSG Field - hwndWindow

hwndWindow ([HWND](#))

The handle for the window associated with the window procedure that is sending the [MMIOM_WINMSG](#) message.

MMIO_WINMSG Field - usMessage

usMessage ([USHORT](#))

The Presentation Manager message number.

MMIO_WINMSG Field - pParam1

pParam1 ([PVOID](#))

The first PM message parameter.

MMIO_WINMSG Field - pParam2

pParam2 ([PVOID](#))

The second PM message parameter.

MMMIDIHEADER

This structure is the MID header and it describes attributes of an open MIDI file. It is the standard presentation format for MIDI data and is returned on `mmioGetHeader` and used on `mmioSetHeader` when `HMMIO` refers to an open MIDI file.

```
typedef struct _MMMIDIHEADER {
    ULONG      ulHeaderLength; /* Header length. */
    ULONG      ulContentType; /* MIDI content. */
    USHORT     usMediaType; /* Media type. */
    MIDIHEADER midiheader; /* Header. */
} MMMIDIHEADER;

typedef MMMIDIHEADER *PMMMIDIHEADER;
```

MMMIDIHEADER Field - ulHeaderLength

ulHeaderLength (`ULONG`)
Length of this header, in bytes.

MMMIDIHEADER Field - ulContentType

ulContentType (`ULONG`)
This field contains the type of MIDI content and it has the following possible values:

MMIO_MIDI_UNKNOWN	Unknown MIDI content.
MMIO_MIDI_VOICE	Limited range.
MMIO_MIDI_MUSIC	FM radio or equivalent.
MMIO_MIDI_HIFI	High quality recording.

MMMIDIHEADER Field - usMediaType

usMediaType (`USHORT`)
The type of multimedia data in the file. This will be determined by MMIO and returned to the application. The currently defined value is:

MMIO_MEDIATYPE_MIDI

MMMIDIHEADER Field - midiheader

midiheader ([MIDIHEADER](#))

A substructure containing additional MIDI information.

MMMOVIEHEADER

This structure is the movie (motion video) header and it describes attributes of an open movie file. It is the standard presentation format for movie data and is returned [mmioGetHeader](#) and used on [mmioSetHeader](#) when [HMMIO](#) refers to an open movie file and the active track for this file is set to MMIO_RESETTRACKS.

```
typedef struct _MMMOVIEHEADER {
    ULONG          ulStructLen;           /* Structure length. */
    ULONG          ulContentType;        /* Movie content type. */
    ULONG          ulMediaType;          /* Movie media type. */
    ULONG          ulMovieCapsFlags;     /* Capabilities. */
    ULONG          ulMaxBytesPerSec;     /* Maximum transfer rate. */
    ULONG          ulPaddingGranularity; /* Pad to a multiple of this value. */
    ULONG          ulSuggestedBufferSize; /* Suggested read buffer size. */
    ULONG          ulStart;              /* Starting time. */
    ULONG          ulLength;             /* Length of file. */
    ULONG          ulNextTrackID;        /* Next identifier. */
    ULONG          ulNumEntries;         /* Track entries. */
    PMMTRACKINFO   pmmTrackInfoList;     /* Pointer to track info. */
    PSZ            pszMovieTitle;        /* ASCIIZ movie title string. */
    ULONG          ulCountry;            /* Country code. */
    ULONG          ulCodepage;           /* Country code page. */
    ULONG          ulAvgBytesPerSec;     /* Average transfer rate. */
} MMMOVIEHEADER;

typedef MMMOVIEHEADER *PMMMOVIEHEADER;
```

MMMOVIEHEADER Field - ulStructLen

ulStructLen ([ULONG](#))

Indicates the length of entire structure.

MMMOVIEHEADER Field - ulContentType

ulContentType ([ULONG](#))

Indicates the movie content type.

MMMOVIEHEADER Field - ulMediaType

ulMediaType (ULONG)
This field contains the movie media type, MMIO_MEDIATYPE_MOVIE.

MMMOVIEHEADER Field - ulMovieCapsFlags

ulMovieCapsFlags (ULONG)
Defines the following capabilities.

MOVIE_HAS_VIDEO
The movie contains video.

MOVIE_HAS_AUDIO
The movie contains audio.

MOVIE_CAN_SEEK
The movie can seek.

MOVIE_CAN_SCAN
The movie can fast scan.

MOVIE_HAS_COPYRIGHT
The movie contains copyrighted data.

MOVIE_WAS_CAPTUREFILE
The movie is a specially allocated file used for capturing real-time video. Applications should warn the user before writing over a file with this flag set because the user probably defragmented this file.

MMMOVIEHEADER Field - ulMaxBytesPerSec

ulMaxBytesPerSec (ULONG)
Maximum transfer rate.

MMMOVIEHEADER Field - ulPaddingGranularity

ulPaddingGranularity (ULONG)
Pad to a multiple of this value.

MMMOVIEHEADER Field - ulSuggestedBufferSize

ulSuggestedBufferSize (ULONG)
Suggested buffer size in bytes for reading the file. If the value specified in this field is greater than the stream buffer specified in the

SPI.INI file for the stream protocol, the MCD will use the *ulSuggestedBufferSize* value as the buffer size for creating streams.

MMMOVIEHEADER Field - ulStart

ulStart ([ULONG](#))

Delay time indicating beginning or starting time of movie. Units defined by *ulRate* and *ulScale*. Normally zero, but delay time can be specified for a stream that does not start concurrently with the movie file.

MMMOVIEHEADER Field - ulLength

ulLength ([ULONG](#))

The length of the movie file.

MMMOVIEHEADER Field - ulNextTrackID

ulNextTrackID ([ULONG](#))

Next available track identifier to add.

MMMOVIEHEADER Field - ulNumEntries

ulNumEntries ([ULONG](#))

Number of track entries in *pmmTrackInfoList*.

MMMOVIEHEADER Field - pmmTrackInfoList

pmmTrackInfoList ([PMMTRACKINFO](#))

Pointer to a track information list. See [MMTRACKINFO](#).

MMMOVIEHEADER Field - pszMovieTitle

pszMovieTitle ([PSZ](#))
Pointer to ASCIIZ movie title string.

MMMOVIEHEADER Field - ulCountry

ulCountry ([ULONG](#))
Country code associated with the movie. Unknown if zero.

MMMOVIEHEADER Field - ulCodepage

ulCodepage ([ULONG](#))
Country code page associated with the movie. Unknown if zero.

MMMOVIEHEADER Field - ulAvgBytesPerSec

ulAvgBytesPerSec ([ULONG](#))
Average transfer rate.

MMMULTITRACKREAD

This structure provides information used with a [MMIOM_MULTITRACKREAD](#) message.

```
typedef struct _MMMULTITRACKREAD {
    ULONG      ulLength;           /* Length of read buffer. */
    PVOID      pBuffer;           /* Pointer to read buffer. */
    ULONG      ulFlags;           /* Read flags. */
    ULONG      ulNumTracks;        /* Number of track entries. */
    PTRACKMAP  pTrackMapList;      /* Pointer to TRACKMAP. */
    ULONG      ulBufferLength;     /* Actual length of read buffer. */
    ULONG      ulReserved;         /* Reserved. */
} MMMULTITRACKREAD;

typedef MMMULTITRACKREAD *PMMMULTITRACKREAD;
```

MMMULTITRACKREAD Field - ulLength

ulLength ([ULONG](#))

Indicates the amount of data in bytes to be read into *pBuffer*. I/O should be performed on this buffer length. The actual buffer length can be bigger and is given in *ulBufferLength*. Video frames can span *pBuffer* + *ulLength* as long as the frame is less than *ulBufferLength* in size.

MMMULTITRACKREAD Field - pBuffer

pBuffer ([PVOID](#))

Pointer to the buffer the contiguous file data will be read into.

MMMULTITRACKREAD Field - ulFlags

ulFlags ([ULONG](#))

MMIOM_MULTITRACKREAD Input flag values:

MULTITRACKREAD_EXTENDED

Indicates the new extended multitrack read structure is passed from caller instead of the previous multitrack read structure.

MMIOM_MULTITRACKREAD Output flag values:

MULTITRACKREAD_NOTDONE

The record table entries of a track are insufficient to hold all the data of the track in *pBuffer*. To continuously get the next record entries, the application must reissue MMIOM_MULTITRACKREAD with the same *pBuffer*.

MULTITRACKREAD_EOF

End of file is reached. This is useful when the number of bytes read do not match the length of the buffer because a record spans into the next buffer.

MMMULTITRACKREAD Field - ulNumTracks

ulNumTracks ([ULONG](#))

Specifies the number of [TRACKMAP](#) entries in *pTrackMapList*.

MMMULTITRACKREAD Field - pTrackMapList

pTrackMapList ([PTRACKMAP](#))

Pointer to a track-to-record list table. See the [TRACKMAP](#) data structure.

MMMULTITRACKREAD Field - ulBufferLength

ulBufferLength ([ULONG](#))
Actual length of read buffer (*pBuffer*).

MMMULTITRACKREAD Field - ulReserved

ulReserved ([ULONG](#))
Reserved for future use and must be set to zero.

MMMULTITRACKWRITE

This structure contains information used with a [MMIOM_MULTITRACKWRITE](#) message.

```
typedef struct _MMMULTITRACKWRITE {  
    ULONG        ulNumTracks;    /* Number of track entries. */  
    PTRACKMAP     pTrackMapList; /* Pointer to TRACKMAP. */  
    ULONG        ulFlags;        /* Flags. */  
    ULONG        ulReserved;     /* Reserved. */  
} MMMULTITRACKWRITE;  
  
typedef MMMULTITRACKWRITE *PMMMULTITRACKWRITE;
```

MMMULTITRACKWRITE Field - ulNumTracks

ulNumTracks ([ULONG](#))
Specifies the number of [TRACKMAP](#) entries in *pTrackMapList*. On output, it returns the actual number of entries written.

MMMULTITRACKWRITE Field - pTrackMapList

pTrackMapList ([PTRACKMAP](#))
Pointer to a track-to-record list table. See the [TRACKMAP](#) data structure.

MMMULTITRACKWRITE Field - ulFlags

ulFlags ([ULONG](#))

Flags.

MULTITRACKWRITE_MERGE

This flag is an input value. MULTITRACKWRITE_MERGE tells the I/O procedure to attempt to interleave the data on the write. The default (without this flag set) is to write all records for each track, then write all records of the next track, and so on.

MMMULTITRACKWRITE Field - ulReserved

ulReserved (ULONG)

Reserved for future use and must be set to zero.

MMTIME

Universal Chinatown time (1/3000 second).

```
typedef ULONG MMTIME;
```

MMTRACKINFO

This structure is a movie subheader and is considered part of a movie header. The *pmmTrackInfoList* field of the [MMMOVIEHEADER](#) structure points to an array of these structures. The *ulNumEntries* field of that structure contains the number of entries in the array. It is part of the standard presentation format for movie data and is returned on [mmioGetHeader](#) and used on [mmioSetHeader](#) when the [HMMIO](#) refers to an open movie file.

```
typedef struct _MMTRACKINFO {
    ULONG    ulTrackID;    /* Track identifier. */
    ULONG    ulMediaType;  /* Media type. */
    ULONG    ulCountry;    /* Country code of track. */
    ULONG    ulCodePage;   /* Country codepage of the track. */
    ULONG    ulReserved;   /* Reserved. */
    ULONG    ulReserved2;  /* Reserved. */
} MMTRACKINFO;

typedef MMTRACKINFO *PMMTRACKINFO;
```

MMTRACKINFO Field - ulTrackID

ulTrackID (ULONG)

Track identifier.

MMTRACKINFO Field - ulMediaType

ulMediaType (ULONG)

The following media types are valid:

MMIO_MEDIATYPE_IMAGE
MMIO_MEDIATYPE_AUDIO
MMIO_MEDIATYPE_MIDI
MMIO_MEDIATYPE_DIGITALVIDEO
MMIO_MEDIATYPE_ANIMATION
MMIO_MEDIATYPE_COMPOUND
MMIO_MEDIATYPE_MOVIE
MMIO_MEDIATYPE_OTHER

Known media type not yet supported.
MMIO_MEDIATYPE_UNKNOWN

Unknown media type.

MMTRACKINFO Field - ulCountry

ulCountry (ULONG)

Country code associated with the track. Unknown if zero.

MMTRACKINFO Field - ulCodePage

ulCodePage (ULONG)

Country code page number associated with the track. Unknown if zero.

MMTRACKINFO Field - ulReserved

ulReserved (ULONG)

Reserved for future use and must be set to zero.

MMTRACKINFO Field - ulReserved2

ulReserved2 (ULONG)

Reserved for future use and must be set to zero.

MMVIDEOCOMPRESS

This structure contains information used on an [MMIOM_COMPRESS](#) and [MMIOM_CODEC_COMPRESS](#) message call to a CODEC. It defines CODEC-specific control information. The *pRunTimeInfo* field of the [MMCOMPRESS](#) structure points to this structure.

```
typedef struct _MMVIDEOCOMPRESS {
    ULONG      ulStructLen; /* Structure length. */
    GENPAL     genpalVideo; /* Video stream palette. */
    PVOID      pControlHdr; /* Control header. */
} MMVIDEOCOMPRESS;
```

```
typedef MMVIDEOCOMPRESS *PMMVIDEOCOMPRESS;
```

MMVIDEOCOMPRESS Field - ulStructLen

ulStructLen ([ULONG](#))

Indicates the length of entire structure.

MMVIDEOCOMPRESS Field - genpalVideo

genpalVideo ([GENPAL](#))

Contains the video stream palette.

MMVIDEOCOMPRESS Field - pControlHdr

pControlHdr ([PVOID](#))

Contains control parameters for compression. The information is CODEC specific, that is, HUFFMAN parameters. If NULL, the default is assumed.

MMVIDEOCODECOMPRESS

This structure contains information used on an [MMIOM_DECOMPRESS](#) and [MMIOM_CODEC_DECOMPRESS](#) message call to a CODEC. It defines CODEC-specific control information. The *pRunTimeInfo* field of the [MMDECOMPRESS](#) structure points to this structure.

```
typedef struct _MMVIDEOCODECOMPRESS {
    ULONG      ulStructLen; /* Structure length. */
    ULONG      ulRectlCount; /* Valid rectangle count. */
    PRECTL     prectl; /* Valid rectangle array. */
    ULONG      ulSkipLength; /* Skipped line length. */
}
```

```
    ULONG      ulDecodeLines; /* Number of lines to decompress. */
    GENPAL     genpalPhysical; /* Physical palette. */
    GENPAL     genpalVideo; /* Video stream palette. */
    RECTL      rectlSrc; /* Source window rectangle. */
    RECTL      rectlDst; /* Destination window rectangle. */
    ULONG      ulDeltaCount; /* Delta count. */
    FOURCC     fccColorSpace; /* Output color space of CODEC. */
    ULONG      fUseScreen; /* Indicator to display engine. */
} MMVIDEODECOMPRESS;

typedef MMVIDEODECOMPRESS *PMMVIDEODECOMPRESS;
```

MMVIDEODECOMPRESS Field - ulStructLen

ulStructLen ([ULONG](#))
Indicates the length of entire structure.

MMVIDEODECOMPRESS Field - ulRectlCount

ulRectlCount ([ULONG](#))
Valid rectangle count for clipping.

MMVIDEODECOMPRESS Field - prectl

prectl ([PRECTL](#))
Valid rectangle array for clipping.

MMVIDEODECOMPRESS Field - ulSkipLength

ulSkipLength ([ULONG](#))
Indicates the skipped line length after each line is decompressed.

MMVIDEODECOMPRESS Field - ulDecodeLines

ulDecodeLines ([ULONG](#))
Indicates number of of lines to be decompressed.

MMVIDEODECOMPRESS Field - genpalPhysical

genpalPhysical ([GENPAL](#))

Contains the physical palette.

MMVIDEODECOMPRESS Field - genpalVideo

genpalVideo ([GENPAL](#))

Contains the video stream palette.

MMVIDEODECOMPRESS Field - rectlSrc

rectlSrc ([RECTL](#))

Contains the source window rectangle.

MMVIDEODECOMPRESS Field - rectlDst

rectlDst ([RECTL](#))

Contains the destination window rectangle.

MMVIDEODECOMPRESS Field - ulDeltaCount

ulDeltaCount ([ULONG](#))

Number of remaining delta frames before the next key or reference frame.

MMVIDEODECOMPRESS Field - fccColorSpace

fccColorSpace ([FOURCC](#))

Allows the output color space of the CODEC to be specified at run time. The *fccColorSpace* is intended to specify one of the following:

- Frame buffer format VRAM (visible).
- Hardware-specific color format (typically located in off-screen VRAM).
- Pointer to system-allocated DIB bitmap.

MMVIDEOCODECOMPRESS Field - fUseScreen

fUseScreen (ULONG)

The display engine uses this parameter to determine whether off-screen hardware is to be used. In the case of recording from a file, this is important; for instance, decoded frames would need to be color converted in software rather than decoded in hardware.

MMVIDEOHEADER

This structure is the video track header and it describes attributes of a video track for an open movie file. It is the standard presentation format for video data and is returned on [mmioGetHeader](#) and used on [mmioSetHeader](#) when the [HMMIO](#) structure refers to an open movie file and the active track for this file is set to a video track.

```
typedef struct _MMVIDEOHEADER {
    ULONG          ulStructLen;          /* Structure length. */
    ULONG          ulContentType;        /* Content type. */
    ULONG          ulMediaType;          /* Video media type. */
    ULONG          ulVideoCapsFlags;     /* Reserved. */
    ULONG          ulWidth;              /* Video width in pels. */
    ULONG          ulHeight;             /* Video height in pels. */
    ULONG          ulScale;              /* Units per sample. */
    ULONG          ulRate;               /* Time units per second. */
    ULONG          ulStart;              /* Video start time. */
    ULONG          ulLength;             /* Length of movie file. */
    ULONG          ulTotalFrames;        /* Number of frames. */
    ULONG          ulInitialFrames;      /* Initial frame for interleaved file. */
    MMTIME         mmtimePerFrame;      /* Frame display time. */
    ULONG          ulSuggestedBufferSize; /* Suggested buffer size. */
    GENPAL         genpalVideo;          /* Pointer to RGB Palette. */
    PMMXDIBHEADER  pmmXDIBHeader;       /* Pointer to MMXDIBHEADER. */
    BOOL           fOpenStretch;         /* Indicator to CODEC. */
    BOOL           fUseScreen;           /* Indicator of hardware-assist. */
} MMVIDEOHEADER;

typedef MMVIDEOHEADER *PMMVIDEOHEADER;
```

MMVIDEOHEADER Field - ulStructLen

ulStructLen (ULONG)

Indicates the length of entire structure.

MMVIDEOHEADER Field - ulContentType

ulContentType (ULONG)
The video content type is either MMIO_VIDEO_DATA or MMIO_VIDEO_UNKNOWN. MMIO_VIDEO_DATA is a recognized video type, MMIO_VIDEO_UNKNOWN is not supported.

MMVIDEOHEADER Field - ulMediaType

ulMediaType (ULONG)
The video media type, MMIO_MEDIATYPE_DIGITALVIDEO.

MMVIDEOHEADER Field - ulVideoCapsFlags

ulVideoCapsFlags (ULONG)
Reserved for future use and must be set to zero.

MMVIDEOHEADER Field - ulWidth

ulWidth (ULONG)
Video width in pels.

MMVIDEOHEADER Field - ulHeight

ulHeight (ULONG)
Video height in pels.

MMVIDEOHEADER Field - ulScale

ulScale (ULONG)
Number of time units per sample.

MMVIDEOHEADER Field - ulRate

ulRate (ULONG)
Number of time units per second.

MMVIDEOHEADER Field - ulStart

ulStart (ULONG)
Specifies start time of the video. Typically this is zero, but can specify a time for a stream that does not start concurrently with the movie file.

MMVIDEOHEADER Field - ulLength

ulLength (ULONG)
The length of the movie file.

MMVIDEOHEADER Field - ulTotalFrames

ulTotalFrames (ULONG)
Total number of video frames.

MMVIDEOHEADER Field - ulInitialFrames

ulInitialFrames (ULONG)
Initial frame for interleaved files. Non-interleaved files should specify zero.

MMVIDEOHEADER Field - mmtimePerFrame

mmtimePerFrame (MMTIME)
Frame display time or OL.

MMVIDEOHEADER Field - ulSuggestedBufferSize

ulSuggestedBufferSize ([ULONG](#))

Suggested buffer size in bytes for reading the file. This field should be set to the maximum frame size for a movie. OS/2 multimedia will use this value to allocate data buffers for playing the movie.

MMVIDEOHEADER Field - genpalVideo

genpalVideo ([GENPAL](#))

An RGB palette table.

MMVIDEOHEADER Field - pmmXDIBHeader

pmmXDIBHeader ([PMMXDIBHEADER](#))

Pointer to the [MMXDIBHEADER](#) structure.

MMVIDEOHEADER Field - fOpenStretch

fOpenStretch ([BOOL](#))

Used to indicate whether the CODEC will be required to decode stretched images.

MMVIDEOHEADER Field - fUseScreen

fUseScreen ([BOOL](#))

Indicates whether off-screen hardware is to be used. In the case of recording from a file, this is important; for instance, decoded frames would need to be color converted in software rather than decoded in hardware.

MMVIDEOOPEN

This structure contains information used on an [MMIOM_CODEC_OPEN](#) message call to a CODEC. It defines data constraint parameters to a video CODEC and is only used if the CODEC_DATA_CONSTRAINT flag is set in the *ulCapsFlags* field of the [CODECINIFILEINFO](#) structure. The *pOtherInfo* field of the [CODECOPEN](#) structure points to this structure.

```
typedef struct _MMVIDEOOPEN {  
    ULONG    ulStructLen;        /* Structure length. */  
    ULONG    ulQuality;          /* Video compression quality. */  
    ULONG    ulKeyFrameRate;     /* Frame rate. */  
};
```

```
ULONG      ulScale;                /* Time units per frame. */
ULONG      ulRate;                 /* Units per second. */
ULONG      ulDataConstraint;       /* Maximum data rate. */
ULONG      ulConstraintInterval;   /* Maximum frame rate. */
} MMVIDEOOPEN;

typedef MMVIDEOOPEN *PMMVIDEOOPEN;
```

MMVIDEOOPEN Field - ulStructLen

ulStructLen (ULONG)
Length of the structure.

MMVIDEOOPEN Field - ulQuality

ulQuality (ULONG)
Indicates the quality of video compression. Range is 0 - 1000. If -1 is specified, the default is assumed. On successful completion, the current quality level is returned.

MMVIDEOOPEN Field - ulKeyFrameRate

ulKeyFrameRate (ULONG)
Indicates key frame rate for compression. For example, 4 means there will be a key frame followed by 3 delta frames. If -1 is specified the default is assumed. On successful completion the current key rate is returned.

MMVIDEOOPEN Field - ulScale

ulScale (ULONG)
Number of time units per frame. If -1 is specified the default is assumed. On successful completion the current scale is returned.

MMVIDEOOPEN Field - ulRate

ulRate (ULONG)
Number of units per second. If -1 is specified the default is assumed. On successful completion the current rate is returned.

MMVIDEOOPEN Field - ulDataConstraint

ulDataConstraint ([ULONG](#))

Maximum number of bytes allowed for the number of frames listed in *ulConstraintInterval*. If -1 is specified the default is assumed. On successful completion the current constrained data rate is returned.

MMVIDEOOPEN Field - ulConstraintInterval

ulConstraintInterval ([ULONG](#))

Number of frames used to constrain the data rate specified in *ulDataConstraint*. If -1 is specified the default is assumed. On successful completion the current constrained frame interval rate is returned.

MMXDIBHEADER

This structure is a subheader for both the image and video header and is usually contained within the [MMIMAGEHEADER](#) or [MMVIDEOHEADER](#) structures. It is part of the standard presentation format for image and video data and is returned on [mmioGetHeader](#) and used on [mmioSetHeader](#) when the [HMMIO](#) refers to an open image or movie file.

```
typedef struct _MMXDIBHEADER {
    XDIBHDR\_PREFIX          XDIBHeaderPrefix; /* PM compatible header extension. */
    BITMAPINFOHEADER2       BMPInfoHeader2;   /* PM bit-map header. */
} MMXDIBHEADER;

typedef MMXDIBHEADER *PMMXDIBHEADER;
```

MMXDIBHEADER Field - XDIBHeaderPrefix

XDIBHeaderPrefix ([XDIBHDR_PREFIX](#))

Extension of OS/2 2.0 PM compatible header.

MMXDIBHEADER Field - BMPInfoHeader2

BMPInfoHeader2 ([BITMAPINFOHEADER2](#))

OS/2 2.0 PM Bit-map Compatibility Header

MMXWAV_HEADER

This structure is a subheader for the audio header and is usually contained within the [MMAUDIOHEADER](#) structure. It contains specific information about the digital audio content including the format of the data representation for the data that it is associated with. It is part of the standard presentation format for audio data and is returned on [mmioGetHeader](#) and used on [mmioSetHeader](#) when [HMMIO](#) refers to an open audio file or open video file when the active track for this file is set to an audio track.

```
typedef struct _MMXWAV_HEADER {
    WAVE\_HEADER           WAVEHeader;      /* Mirrors RIFF wave header info. */
    XWAV\_HEADERINFO       XWAVHeaderInfo;  /* Additional audio parameters. */
} MMXWAV_HEADER;

typedef MMXWAV_HEADER *PMMXWAV_HEADER;
```

MMXWAV_HEADER Field - WAVEHeader

WAVEHeader ([WAVE_HEADER](#))

Wave header information as explained under WAVE Header. This field maps to the RIFF wave header.

MMXWAV_HEADER Field - XWAVHeaderInfo

XWAVHeaderInfo ([XWAV_HEADERINFO](#))

Includes additional audio parameters.

MPARAM

A 4-byte message-dependent parameter structure.

```
typedef VOID *MPARAM;
```

Certain elements of information, placed into the parameters of a message, have data types that do not use all four bytes of this data type. The rules governing these cases are:

BOOL	The value is contained in the low word and the high word is 0.
SHORT	The value is contained in the low word and its sign is extended into the high word.
USHORT	The value is contained in the low word and the high word is 0.
NULL	The entire four bytes are 0.

The structure of this data type depends on the message. For details, see the description of the particular message.

MRESULT

A 4-byte message-dependent reply parameter structure.

```
typedef VOID *MRESULT;
```

Certain elements of information, placed into the parameters of a message, have data types that do not use all four bytes of this data type. The rules governing these cases are:

BOOL	The value is contained in the low word and the high word is 0.
SHORT	The value is contained in the low word and its sign is extended into the high word.
USHORT	The value is contained in the low word and the high word is 0.
NULL	The entire four bytes are 0.

The structure of this data type depends on the message. For details, see the description of the particular message.

MSG_COMMON

This structure contains fields common to the [SpiSendMsg](#) function.

```
typedef struct _MSG_COMMON {
    ULONG    ulMsgLen; /* Length of structure. */
} MSG_COMMON;
```

MSG_COMMON Field - ulMsgLen

ulMsgLen (ULONG)

The first entry in the MSG_COMMON data structure must be the length of the structure.

MSG_REPORTEVENT

This structure contains fields for the SHC_REPORT_EVENT message (passed with [SpiSendMsg](#)).

```
typedef struct _MSG_REPORTEVENT {
    ULONG    ulMsgLen; /* Length of structure. */
    HEVENT   hevent; /* Event handle. */
    ULONG    ulStreamTime; /* Time in milliseconds of stream position. */
} MSG_REPORTEVENT;

typedef MSG_REPORTEVENT FAR *PMSG_REPORTEVENT;
```

MSG_REPORTEVENT Field - ulMsgLen

ulMsgLen (ULONG)

Length of structure.

MSG_REPORTEVENT Field - hevent

hevent ([HEVENT](#))

Event handle passed back to stream handler.

MSG_REPORTEVENT Field - ulStreamTime

ulStreamTime ([ULONG](#))

Time in milliseconds of stream position.

MSG_REPORTINT

This structure contains fields for the SHC_REPORT_INT message (passed with [SpiSendMsg](#)).

```
typedef struct _MSG_REPORTINT {
    ULONG    ulMsgLen;        /* Length of structure. */
    PVOID    pBuffer;        /* Return pointer. */
    ULONG    ulFlag;         /* Reason for interrupt. */
    ULONG    ulStatus;       /* Bytes read or written. */
    ULONG    ulStreamTime;   /* Stream time in milliseconds. */
} MSG_REPORTINT;

typedef MSG_REPORTINT FAR *PMSG_REPORTINT;
```

MSG_REPORTINT Field - ulMsgLen

ulMsgLen ([ULONG](#))

Length of structure.

MSG_REPORTINT Field - pBuffer

pBuffer ([PVOID](#))

Pointer to buffer being returned (or null for no buffer to return).

MSG_REPORTINT Field - ulFlag

ulFlag (ULONG)

Specifies the reason for the interrupt. This field defines the following flags:

```
ERROR
STREAM_STOP_NOW
SHD_READ_COMPLETE
SHD_WRITE_COMPLETE
```

Note:

1. ERROR must be ORed with SHD_READ_COMPLETE or SHD_WRITE_COMPLETE.
2. Do not set SHD_READ_COMPLETE or SHD_WRITE_COMPLETE if not returning a buffer.

MSG_REPORTINT Field - ulStatus

ulStatus (ULONG)

Return code or bytes read or written.

MSG_REPORTINT Field - ulStreamTime

ulStreamTime (ULONG)

Stream time in milliseconds.

OVRU_EVCB

This structure describes a synchronization overrun event. The event can be enabled through an [SpiEnableEvent](#) call.

```
typedef struct _OVRU_EVCB {
    ULONG      ulType;           /* Event type. */
    ULONG      ulSubType;        /* Event subtype. */
    ULONG      ulFlags;          /* Flags. */
    HSTREAM    hstream;          /* Stream handle. */
    HID        hid;              /* Stream handler ID. */
    ULONG      ulStatus;         /* Event status. */
    MMTIME     mmtimeSlave;      /* Slave stream time. */
    MMTIME     mmtimeStart;      /* Start start offset. */
    MMTIME     mmtimeMaster;     /* Master stream time. */
} OVRU_EVCB;

typedef OVRU_EVCB *POVRU_EVCB;
```


OVRU_EVCB Field - ulType

ulType (ULONG)
Identifies the event type. For this event, the type must be EVENT_SYNCOVERRUN.

OVRU_EVCB Field - ulSubType

ulSubType (ULONG)
Event subtype.

OVRU_EVCB Field - ulFlags

ulFlags (ULONG)
Set of bit flags with various meanings.

OVRU_EVCB Field - hstream

hstream (HSTREAM)
Handle that identifies the stream instance for this event. This field must be filled in by the application media control device for the [SpiEnableEvent](#) call when this event is enabled. It also must be filled in by the stream handler when reporting this event.

OVRU_EVCB Field - hid

hid (HID)
ID that identifies the stream handler that reported this event. This field can be filled in by the application media control device for the [SpiEnableEvent](#) call when this event is enabled. In this case, it identifies which stream handler must report this event. The stream handler must be able to report this event for the [SpiEnableEvent](#) call to succeed. This field also must be filled in by the stream handler when reporting this event.

OVRU_EVCB Field - ulStatus

ulStatus (ULONG)

Status of the event. This field is not used.

OVRU_EVCB Field - mmtimeSlave

mmtimeSlave ([MMTIME](#))

Contains the current slave stream time at the point at which the synchronization overrun occurred.

OVRU_EVCB Field - mmtimeStart

mmtimeStart ([MMTIME](#))

Contains the time that the slave stream was started relative to the start of the master stream.

OVRU_EVCB Field - mmtimeMaster

mmtimeMaster ([MMTIME](#))

Contains the current master stream time at the point at which the synchronization overrun occurred.

PARM_ASSOC

This structure contains fields for the [SHC_ASSOCIATE](#) message.

```
typedef struct _PARM_ASSOC {
    ULONG      ulFunction; /* Handler command function. */
    HID        hid;        /* Handler ID. */
    HSTREAM    hstream;    /* Stream handle instance. */
    PACB       pacb;       /* Pointer to associate control block. */
} PARM_ASSOC;
```

```
typedef PARM_ASSOC FAR *PPARM_ASSOC;
```

PARM_ASSOC Field - ulFunction

ulFunction ([ULONG](#))

Handler command function.

PARM_ASSOC Field - hid

hid (HID)
Handler ID.

PARM_ASSOC Field - hstream

hstream (HSTREAM)
Stream handle instance.

PARM_ASSOC Field - pacb

pacb (PACB)
Points to ACB.

PARM_CLOSE

This structure contains fields for the SHC_CLOSE message.

```
typedef struct _PARM_CLOSE {
    ULONG      ulFunction; /* Handler command function. */
    HID        hid;        /* Handler ID. */
} PARM_CLOSE;

typedef PARM_CLOSE FAR *PPARM_CLOSE;
```

PARM_CLOSE Field - ulFunction

ulFunction (ULONG)
Handler command function.

PARM_CLOSE Field - hid

hid (HID)
Handler ID.

PARM_CREATE

This structure contains fields for the SHC_CREATE message.

```
typedef struct _PARM_CREATE {
    ULONG      ulFunction; /* Handler command function. */
    HID        hid;        /* Handler ID. */
    HSTREAM    hstream;    /* Handle of stream instance. */
    SPCBKEY    spcbkey;    /* Type of stream to create. */
    PSPCB      pspcb;      /* Pointer from handler. */
    HSTREAM    hstreamBuf; /* Used for split streams. */
    PDCB       pdcB;       /* Pointer to device control block. */
} PARM_CREATE;

typedef PARM_CREATE FAR *PPARM_CREATE;
```

PARM_CREATE Field - ulFunction

ulFunction (ULONG)
Handler command function.

PARM_CREATE Field - hid

hid (HID)
Handler ID.

PARM_CREATE Field - hstream

hstream (HSTREAM)
Handle of stream instance.

PARM_CREATE Field - spcbkey

spcbkey (SPCBKEY)

Type of stream to create.

PARM_CREATE Field - pspcb

pspcb ([PSPCB](#))
Points to [SPCB](#) from the handler.

PARM_CREATE Field - hstreamBuf

hstreamBuf ([HSTREAM](#))
Is used for split streams or to associate another stream with this stream.

PARM_CREATE Field - pdcb

pdcb ([PDCB](#))
Points to [DCB](#).

PARM_DEREG

This structure contains fields for the [SMH_DEREGISTER](#) message.

```
typedef struct _PARM_DEREG {  
    ULONG      ulFunction; /* SMH command function. */  
    PSZ        pszSHName; /* Maximum size. */  
} PARM_DEREG;  
  
typedef PARM_DEREG FAR *PPARM_DEREG;
```

PARM_DEREG Field - ulFunction

ulFunction ([ULONG](#))
SMH command function.

PARM_DEREG Field - pszSHName

pszSHName ([PSZ](#))

Maximum size for the name is 8 characters, or 9 if including a null.

PARM_DESTROY

This structure contains fields for the [SHC_DESTROY](#) message.

```
typedef struct _PARM_DESTROY {  
    ULONG      ulFunction; /* Handler command function. */  
    HID        hid;        /* Handler ID. */  
    HSTREAM     hstream;    /* Handle of stream instance. */  
} PARM_DESTROY;  
  
typedef PARM_DESTROY FAR *PPARM_DESTROY;
```

PARM_DESTROY Field - ulFunction

ulFunction ([ULONG](#))

Handler command function.

PARM_DESTROY Field - hid

hid ([HID](#))

Handler ID.

PARM_DESTROY Field - hstream

hstream ([HSTREAM](#))

Handle of stream instance.

PARM_DISEVENT

This structure contains fields for the [SHC_DISABLE_EVENT](#) message.

```
typedef struct _PARM_DISEVENT {
```

```
    ULONG      ulFunction; /* Handler command function. */
    HID        hid;        /* Handler ID. */
    HSTREAM     hstream;    /* Handle to stream for this event. */
    HEVENT      hevent;     /* Handle of event to disable. */
} PARM_DISEVENT;

typedef PARM_DISEVENT FAR *PPARM_DISEVENT;
```

PARM_DISEVENT Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_DISEVENT Field - hid

hid ([HID](#))
Handler ID.

PARM_DISEVENT Field - hstream

hstream ([HSTREAM](#))
Handle to stream for this event.

PARM_DISEVENT Field - hevent

hevent ([HEVENT](#))
Handle of event to disable.

PARM_DISSYNC

This structure contains fields for the [SHC_DISABLE_SYNC](#) message.

```
typedef struct _PARM_DISSYNC {
    ULONG      ulFunction; /* Handler command function. */
    HID        hid;        /* Handler ID. */
    HSTREAM     hStream;    /* Handle of stream instance. */
} PARM_DISSYNC;
```

```
typedef PARM_DISSYNC FAR *PPARM_DISSYNC;
```

PARM_DISSYNC Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_DISSYNC Field - hid

hid ([HID](#))
Handler ID.

PARM_DISSYNC Field - hStream

hStream ([HSTREAM](#))
Handle of stream instance.

PARM_ENEVENT

This structure contains fields for the [SHC_ENABLE_EVENT](#) message.

```
typedef struct _PARM_ENEVENT {  
    ULONG      ulFunction; /* Handler command function. */  
    HID        hid;        /* Handler ID. */  
    HEVENT     hevent;     /* Handle of events to enable. */  
    PEVCB      pevcbUser;  /* User event information. */  
} PARM_ENEVENT;  
  
typedef PARM_ENEVENT FAR *PPARM_ENEVENT;
```

PARM_ENEVENT Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_ENEVENT Field - hid

hid ([HID](#))
Handler ID.

PARM_ENEVENT Field - hevent

hevent ([HEVENT](#))
Handle of events to enable.

PARM_ENEVENT Field - pevcbUser

pevcbUser ([PEVCB](#))
Gives you information about user events, such as *hstream*, *EventType*, and *EventSubType*.

PARM_ENOTIFY

This structure contains fields for the [SMH_NOTIFY](#) message.

```
typedef struct _PARM_ENOTIFY {  
    ULONG      ulFunction;    /* Input - SMH command function. */  
    HID        hid;          /* Input - Handler ID. */  
    HSTREAM    hstream;      /* Input - Stream instance handle. */  
    ULONG      ulFlags;       /* Input/Output - Flags. */  
    ULONG      ulGetNumEntries; /* Input/Output - # of table entries. */  
    ULONG      ulRetNumEntries; /* Input/Output - # of table entries. */  
    PVOID      pGetBufTab;    /* Output - Pointer to buffer/record table. */  
    PVOID      pRetBufTab;    /* Input - Pointer to buffer/record table. */  
    ULONG      ulParm1;       /* Reserved for future use. */  
    ULONG      ulParm2;       /* Reserved for future use. */  
} PARM_ENOTIFY;  
  
typedef PARM_ENOTIFY FAR *PPARM_ENOTIFY;
```

PARM_ENOTIFY Field - ulFunction

ulFunction ([ULONG](#))
Input - SMH command function.

PARM_ENOTIFY Field - hid

hid (HID)
Input - Handler ID.

PARM_ENOTIFY Field - hstream

hstream (HSTREAM)
Input - Stream instance handle.

PARM_ENOTIFY Field - ulFlags

ulFlags (ULONG)
The following flags can be used:

- BUF_GETEMPTY

Pointer returned in SMH_pGetBuffer. Get one or more of the empty buffers to fill.
- BUF_RETURNFULL

Pointer passed in SMH_pRetBuffer. Return one or more filled records.
- BUF_GET_FULL

Pointer returned in SMH_pGetBuffer.
- BUF_RETURNEMPTY

Pointer passed in SMH_pRetBuffer
- BUF_GIVEBUF

Pointer passed in SMH_pRetBuffer. Give a user-provided buffer to SSM. Refer to the bit flag in the SPCB used to indicate that user-provided buffers are being used (ring 3 only). Buffer pointers are passed in *pRetBufTab*.
- BUF_EOS

End of stream (last buffer produced). This must be used with BUF_RETURN_FULL and BUF_GIVEBUF. This flag is passed to the target stream handler with the last buffer in the stream.
- BUF_LINEAR

Pointers are global linear (Ring 0 only).
- BUF_RECORDS

Pointers are global linear (Ring 0 only). Return one or more filled records.
- BUF_LASTRECORD

Indicates that the last record in a buffer is filled. This can be used with the BUF_RETURNFULL + BUF_RECORDS combination only.
- BUF_LINEAR

Pointers are global linear (Ring 0 only).
- BUF_PHYSICAL

	Pointers are physical (Ring 0 only). This is valid for physically contiguous buffers.
BUF_RESERVED	Reserved.
BUF_EXTENDED	This bit must be set if you are using extended data structures (PARM_ENOTIFY).
BUF_GDT	Pointers are GDT sel:offset
BUF_EXTENDEDPTR	Use extended structures, EPTGTBUFTAB and EPSRCBUFTAB (implies BUF_EXTENDED).

PARM_ENOTIFY Field - ulGetNumEntries

ulGetNumEntries ([ULONG](#))
Input/Output - # of table entries.

PARM_ENOTIFY Field - ulRetNumEntries

ulRetNumEntries ([ULONG](#))
Input/Output - # of table entries.

PARM_ENOTIFY Field - pGetBufTab

pGetBufTab ([PVOID](#))
Pointer to extended structure, [ESRCBUFTAB](#).

PARM_ENOTIFY Field - pRetBufTab

pRetBufTab ([PVOID](#))
Pointer to extended structure, [ETGTBUFTAB](#).

PARM_ENOTIFY Field - ulParm1

ulParm1 ([ULONG](#))
Reserved for future use.

PARM_ENOTIFY Field - ulParm2

ulParm2 ([ULONG](#))
Reserved for future use.

PARM_ENSYNC

This structure contains fields for the [SHC_ENABLE_SYNC](#) message.

```
typedef struct _PARM_ENSYNC {
    ULONG      ulFunction; /* Handler command function. */
    HID        hid;        /* Handler ID. */
    HSTREAM    hstream;    /* Stream handle instance. */
    ULONG      ulFlags;     /* Sync flags. */
    MMTIME     mmtimeSync; /* Granularity. */
    PSYNC\_EVCB pevcbSyncPulse; /* Sync pulse info. */
    ULONG      ulSyncPulseSem; /* 16 bit semaphore. */
} PARM_ENSYNC;

typedef PARM_ENSYNC FAR *PPARM_ENSYNC;
```

PARM_ENSYNC Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_ENSYNC Field - hid

hid ([HID](#))
Handler ID.

PARM_ENSYNC Field - hstream

hstream ([HSTREAM](#))

Stream handle instance.

PARM_ENSYNC Field - ulFlags

ulFlags ([ULONG](#))

Stop flags. The following flags are available:

SYNC_MASTER

The handler will be a master for this sync.

SYNC_SLAVE

The handler will be a slave for this sync.

PARM_ENSYNC Field - mmtimeSync

mmtimeSync ([MMTIME](#))

Granularity of sync interval. If NULL, the default is used.

PARM_ENSYNC Field - pevcbSyncPulse

pevcbSyncPulse ([PSYNC_EVCB](#))

Sync pulse EVCB information.

PARM_ENSYNC Field - ulSyncPulseSem

ulSyncPulseSem ([ULONG](#))

Optional 16-bit system semaphore for handler.

PARM_ENUMPROT

This structure contains fields for the [SHC_ENUMERATE_PROTOCOLS](#) message.

```
typedef struct _PARM_ENUMPROT {
    ULONG      ulFunction;      /* Handler command function. */
    HID        hid;             /* Handler ID. */
    PVOID      paSPCBKeys;      /* Buffer pointer. */
    PULONG     pulNumSPCBKeys;   /* Number of buffer entries. */
} PARM_ENUMPROT;
```

```
typedef PARM_ENUMPROT FAR *PPARM_ENUMPROT;
```

PARM_ENUMPROT Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_ENUMPROT Field - hid

hid ([HID](#))
Handler ID.

PARM_ENUMPROT Field - paSPCBKeys

paSPCBKeys ([PVOID](#))
Pointer to buffer to fill with SPCB keys. See [SPCBKEY](#).

PARM_ENUMPROT Field - pulNumSPCBKeys

pulNumSPCBKeys ([PULONG](#))
The number of buffer entries on input; the number of SPCB keys on output.

PARM_EVENT

This structure contains fields for the [SMH_REPORTEVENT](#) message.

```
typedef struct _PARM_EVENT {  
    ULONG      ulFunction; /* SMH command function. */  
    HID        hid;        /* Handler ID. */  
    HEVENT     hevent;      /* Explicit events only. */  
    PEVCB      pevcbEvent;  /* Event status and HSTREAM. */  
} PARM_EVENT;  
  
typedef PARM_EVENT FAR *PPARM_EVENT;
```

PARM_EVENT Field - ulFunction

ulFunction ([ULONG](#))
SMH command function.

PARM_EVENT Field - hid

hid ([HID](#))
Handler ID.

PARM_EVENT Field - hevent

hevent ([HEVENT](#))
Used only for explicit event. Must be 0 for sync and implicit events (such as error events).

PARM_EVENT Field - pevcbEvent

pevcbEvent ([PEVCB](#))
Event status and HSTREAM.

PARM_GPROT

This structure contains fields for the [SHC_GET_PROTOCOL](#) message.

```
typedef struct _PARM_GPROT {  
    ULONG      ulFunction; /* Handler command function. */  
    HID        hid;        /* Handler ID. */  
    SPCBKEY    spcbkey;    /* Key of SPCB. */  
    PSPCB      pspcb;      /* Pointer to SPCB. */  
} PARM_GPROT;  
  
typedef PARM_GPROT FAR *PPARM_GPROT;
```

PARM_GPROT Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_GPROT Field - hid

hid ([HID](#))
Handler ID.

PARM_GPROT Field - spcbkey

spcbkey ([SPCBKEY](#))
Key of SPCB.

PARM_GPROT Field - pspcb

pspcb ([PSPCB](#))
See [SPCB](#).

PARM_GTIME

This structure contains fields for the [SHC_GET_TIME](#) message.

```
typedef struct _PARM_GTIME {
    ULONG      ulFunction; /* Handler command function. */
    HID        hid;        /* Handler ID. */
    HSTREAM    hstream;    /* Handle of stream instance. */
    MMTIME     mmtimeCurrent; /* Returns current stream time. */
} PARM_GTIME;

typedef PARM_GTIME FAR *PPARM_GTIME;
```

PARM_GTIME Field - ulFunction

ulFunction ([ULONG](#))

Handler command function.

PARM_GTIME Field - hid

hid ([HID](#))
Handler ID.

PARM_GTIME Field - hstream

hstream ([HSTREAM](#))
Handle of stream instance.

PARM_GTIME Field - mmtimeCurrent

mmtimeCurrent ([MMTIME](#))
Returns current stream time.

PARM_INSTPROT

This structure contains fields for the [SHC_INSTALL_PROTOCOL](#) message.

```
typedef struct _PARM_INSTPROT {  
    ULONG      ulFunction; /* Handler command function. */  
    HID        hid;        /* Handler ID. */  
    SPCBKEY    spcbkey;    /* Key of SPCB. */  
    PSPCB      pspcb;      /* Pointer to SPCB to install. */  
    ULONG      ulFlags;    /* Install/deinstall flags. */  
} PARM_INSTPROT;  
  
typedef PARM_INSTPROT FAR *PPARM_INSTPROT;
```

PARM_INSTPROT Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_INSTPROT Field - hid

hid ([HID](#))
Handler ID.

PARM_INSTPROT Field - spcbkey

spcbkey ([SPCBKEY](#))
Key of SPCB.

PARM_INSTPROT Field - pspcb

pspcb ([PSPCB](#))
See [SPCB](#).

PARM_INSTPROT Field - ulFlags

ulFlags ([ULONG](#))
Install/deinstall flags.

PARM_LOCKM

This structure contains fields for the [SMH_LOCKMEM](#) message.

```
typedef struct _PARM_LOCKM {  
    ULONG      ulFunction; /* SMH command function. */  
    PSZ        pBuffer;    /* Linear address to lock. */  
    ULONG      ulBufSize;  /* Size of memory to lock. */  
    PLOCKH     plockh;     /* Lock handle. */  
    ULONG      ulFlags;    /* Function requested. */  
} PARM_LOCKM;  
  
typedef PARM_LOCKM FAR *PPARM_LOCKM;
```

PARM_LOCKM Field - ulFunction

ulFunction (ULONG)
SMH command function.

PARM_LOCKM Field - pBuffer

pBuffer (PSZ)
The maximum size is equal to 8 characters, or 9 including a null.

PARM_LOCKM Field - ulBufSize

ulBufSize (ULONG)
Size of memory to lock.

PARM_LOCKM Field - plockh

plockh (PLOCKH)
Lock handle.

PARM_LOCKM Field - ulFlags

ulFlags (ULONG)
The following flags can be used:

SSM_LOCKMEM	Lock (fix) memory
SSM_UNLOCKMEM	Unlock memory
SSM_CONTIGLOCK	Lock memory contiguously

PARM_NEGOTIATE

This structure contains fields for the SHC_NEGOTIATE_RESULT message.

```
typedef struct _PARM_NEGOTIATE {
    ULONG      ulFunction; /* Handler command function. */
    HID        hid;        /* Handler ID. */
    HSTREAM    hstream;    /* Handle of stream instance. */
    PSPCB      pspcb;      /* Pointer to negotiated SPCB. */
    ULONG      ulErrorStatus; /* Error status. */
} PARM_NEGOTIATE;

typedef PARM_NEGOTIATE FAR *PPARM_NEGOTIATE;
```

PARM_NEGOTIATE Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_NEGOTIATE Field - hid

hid ([HID](#))
Handler ID.

PARM_NEGOTIATE Field - hstream

hstream ([HSTREAM](#))
Handle of stream instance.

PARM_NEGOTIATE Field - pspcb

pspcb ([PSPCB](#))
Points to negotiated [SPCB](#).

PARM_NEGOTIATE Field - ulErrorStatus

ulErrorStatus ([ULONG](#))
On error, indicates field in the SPCB failed the negotiation.

PARM_NOTIFY

This structure contains fields for the [SMH_NOTIFY](#) message.

```
typedef struct _PARM_NOTIFY {  
    ULONG      ulFunction;      /* SMH command function. */  
    HID        hid;            /* Handler ID. */  
    HSTREAM     hstream;        /* Stream instance handle. */  
    ULONG      ulFlags;         /* Flags. */  
    ULONG      ulGetNumEntries; /* Number of table entries. */  
    ULONG      ulRetNumEntries; /* Number of table entries. */  
    PVOID      pGetBufTab;      /* Output - Pointer to buffer/record table. */  
    PVOID      pRetBufTab;      /* Input - Pointer to buffer/record table. */  
} PARM_NOTIFY;  
  
typedef PARM_NOTIFY FAR *PPARM_NOTIFY;
```

PARM_NOTIFY Field - ulFunction

ulFunction ([ULONG](#))
SMH command function.

PARM_NOTIFY Field - hid

hid ([HID](#))
Handler ID.

PARM_NOTIFY Field - hstream

hstream ([HSTREAM](#))
Stream instance handle.

PARM_NOTIFY Field - ulFlags

ulFlags ([ULONG](#))
The following flags can be used:

BUF_GETEMPTY

	Pointer returned in SMH_pGetBuffer. Get one or more empty buffers to fill.
BUF_RETURNFULL	Pointer passed in SMH_pRetBuffer. Return one or more filled buffers.
BUF_GET_FULL	Pointer returned in SMH_pGetBuffer. Get one or more filled buffers to use.
BUF_RETURNEMPTY	Pointer passed in SMH_pRetBuffer. Return one or more empty (use) buffers.
BUF_GIVEBUF	Pointer passed in SMH_pRetBuffer. Give a user-provided buffer to SSM. Refer to the bit flag in the SPCB used to indicate that user-provided buffers are being used (ring 3 only). Buffers are passed in <i>pRetBufTab</i> .
BUF_EOS	Indicates end-of-stream (last buffer produced). This must be used with BUF_RETURNFULL and BUF_GIVEBUF. This flag is passed to the target stream handler with the last buffer in the stream.
BUF_LINEAR	Pointers are global linear (Ring 0 only).
BUF_RECORDS	Pointers are global linear (Ring 0 only).
BUF_LASTRECORD	(BUF_RETURNFULL only). Records are marked as being the last in the buffer. This tells SSM the maximum number of records produced for this buffer.
BUF_LINEAR	Pointers are global linear (Ring 0 only).
BUF_PHYSICAL	Pointers are physical (Ring 0 only). This is valid for physically contiguous buffers.
BUF_RESERVED	Reserved.
BUF_EXTENDED	Set this flag if you are using extended SMH_NOTIFY structures.
BUF_GDT	Pointers are GDT sel:offset.
BUF_EXTENDEDPTR	Use extended pointer SMH_NOTIFY structures, EPTGTBUFTAB and EPSRCBUFTAB (implies BUF_EXTENDED).

PARM_NOTIFY Field - ulGetNumEntries

ulGetNumEntries ([ULONG](#))
Number of table entries.

PARM_NOTIFY Field - ulRetNumEntries

ulRetNumEntries ([ULONG](#))
Number of table entries.

PARM_NOTIFY Field - pGetBufTab

pGetBufTab ([PVOID](#))

Output - Pointer to buffer/record table.

PARM_NOTIFY Field - pRetBufTab

pRetBufTab ([PVOID](#))

Input - Pointer to buffer/record table.

PARM_REG

This structure contains fields for the [SMH_REGISTER](#) message.

```
typedef struct _PARM_REG {  
    ULONG        ulFunction;           /* SMH command function. */  
    PSZ          pszSHName;           /* Maximum size. */  
    HID FAR      *phidSrc;            /* Handler ID of source. */  
    HID FAR      *phidTgt;            /* Handler ID of target. */  
    PSHCFN       pshcfnEntry;         /* Handler entry point. */  
    ULONG        ulFlags;              /* Flags. */  
    ULONG        ulMaxNumStreams;      /* Maximum number of streams. */  
    ULONG        ulMaxNumEvents;      /* Maximum number of events. */  
} PARM_REG;
```

```
typedef PARM_REG FAR *PPARM_REG;
```

PARM_REG Field - ulFunction

ulFunction ([ULONG](#))

SMH command function.

PARM_REG Field - pszSHName

pszSHName ([PSZ](#))

Maximum size is 8 characters, or 9 if including a null.

PARM_REG Field - phidSrc

phidSrc ([HID FAR *](#))
 Device driver stream handlers cannot use the IDC interface to the Sync/Stream Manager during initialization. It must use the IOCTL function to perform the SMH_REGISTER function. Addresses are 16:16 for this IOCTL because it comes from a device driver.

PARM_REG Field - phidTgt

phidTgt ([HID FAR *](#))
 Device driver stream handlers cannot use the IDC interface to the Sync/Stream Manager during initialization. It must use the IOCTL function to perform the SMH_REGISTER function. Addresses are 16:16 for this IOCTL because it comes from a device driver.

PARM_REG Field - pshcfnEntry

pshcfnEntry ([PSHCFN](#))
 Handler entry point.

PARM_REG Field - ulFlags

ulFlags ([ULONG](#))
 The following flags are used:

REGISTER_TGT_HNDLR

This handler is Target. Handler could be either source or target.

REGISTER_SRC_HNDLR

This handler is Source. Handler could be either source or target.

REGISTER_NONSTREAMING

Handler is non-streaming.

VALIDREGISTERFLAGS (REGISTER_TGT_HNDLR | REGISTER_SRC_HNDLR | REGISTER_NONSTREAMING)

SSMDD_CATEGORY

Category for ring 0 stream handler.

SMH_DEREGISTER

Function for ring 0 stream handler.

PARM_REG Field - ulMaxNumStreams

ulMaxNumStreams ([ULONG](#))
Maximum number of streams.

PARM_REG Field - ulMaxNumEvents

ulMaxNumEvents ([ULONG](#))
Maximum number of events.

PARM_SEEK

This structure contains fields for the [SHC_SEEK](#) message.

```
typedef struct _PARM_SEEK {  
    ULONG      ulFunction; /* Handler command function. */  
    HID        hid;        /* Handler ID. */  
    HSTREAM    hstream;    /* Handle of event to enable. */  
    ULONG      ulFlags;     /* Seek flags. */  
    LONG       lSeekPoint;  /* Seek to point, MMTIME or other. */  
} PARM_SEEK;  
  
typedef PARM_SEEK FAR *PPARM_SEEK;
```

PARM_SEEK Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_SEEK Field - hid

hid ([HID](#))
Handler ID.

PARM_SEEK Field - hstream

hstream ([HSTREAM](#))
Handle of event to enable.

PARM_SEEK Field - ulFlags

ulFlags ([ULONG](#))
Seek flags. See *ulFlags* for [SpiSeekStream](#).

PARM_SEEK Field - ISeekPoint

ISeekPoint ([LONG](#))
Seek to point, MMTIME or other.

PARM_SNDMSG

This structure contains fields for the [SHC_SENDMSG](#) message.

```
typedef struct _PARM_SNDMSG {  
    ULONG      ulFunction; /* Handler command function. */  
    HID        hid;        /* Handler ID. */  
    HSTREAM    hstream;    /* Handle of stream instance. */  
    ULONG      ulMsgType;   /* Stream handler message type. */  
    PVOID      pMsg;        /* Pointer to message control block. */  
} PARM_SNDMSG;  
  
typedef PARM_SNDMSG FAR *PPARM_SNDMSG;
```

PARM_SNDMSG Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_SNDMSG Field - hid

hid ([HID](#))
Handler ID.

PARM_SNDMSG Field - hstream

hstream ([HSTREAM](#))
Handle of stream instance.

PARM_SNDMSG Field - ulMsgType

ulMsgType ([ULONG](#))
Stream handler message type.

PARM_SNDMSG Field - pMsg

pMsg ([PVOID](#))
See the [SHC_SENDMSG](#) message control block.

PARM_START

This structure contains fields for the [SHC_START](#) message.

```
typedef struct _PARM_START {  
    ULONG      ulFunction; /* Handler command function. */  
    HID        hid;        /* Handler ID. */  
    HSTREAM    hstream;    /* Handle of stream instance. */  
    ULONG      ulFlags;     /* Start flag. */  
} PARM_START;  
  
typedef PARM_START FAR *PPARM_START;
```

PARM_START Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_START Field - hid

hid ([HID](#))
Handler ID.

PARM_START Field - hstream

hstream ([HSTREAM](#))
Handle of stream instance.

PARM_START Field - ulFlags

ulFlags ([ULONG](#))
The following flag can be used:

FLUSH_START
This start is a result of a Flush Stop command.

PARM_STOP

This structure contains fields for the [SHC_STOP](#) message.

```
typedef struct _PARM_STOP {  
    ULONG      ulFunction; /* Handler command function. */  
    HID        hid;        /* Handler ID. */  
    HSTREAM     hstream;    /* Handle of stream instance. */  
    ULONG      ulFlags;     /* Stop flags. */  
} PARM_STOP;  
  
typedef PARM_STOP FAR *PPARM_STOP;
```

PARM_STOP Field - ulFunction

ulFunction ([ULONG](#))
Handler command function.

PARM_STOP Field - hid

hid ([HID](#))
Handler ID.

PARM_STOP Field - hstream

hstream ([HSTREAM](#))
Handle of stream instance.

PARM_STOP Field - ulFlags

ulFlags ([ULONG](#))
For stop flag information, see *ulFlags* for [SpiStopStream](#). These include:

SPI_STOP_STREAM
SPI_STOP_SLAVES
SPI_STOP_FLUSH
SPI_STOP_DISCARD

PCODECPROC

Pointer to a CODEC procedure entry point.

```
typedef CODECPROC FAR *PCODECPROC;
```

All CODEC messages are passed to a CODEC using this interface. The syntax for a direct CODEC procedure call is shown below:

```
typedef LONG (APIENTRY CODECPROC)  
            (PHCODEC phcodec,  
             USHORT usMsg,  
             LONG lParam1,  
             LONG lParam2);
```

PEVFN

Pointer to an application-event entry routine. This routine is used to handle stream events.

```
typedef EVFN (FAR *PEVFN);
```

In the header file, this is a two-part definition as shown below:

```
typedef ULONG (APIENTRY EVFN) (PEVCB pevcb);
typedef EVFN (FAR *PEVFN);
```

PFN

Pointer to a procedure.

```
typedef _PFN *PFN;
```

In the header file, this is a two-part definition as shown below:

```
typedef int (APIENTRY _PFN) ();
typedef _PFN *PFN;
```

PFNMIDI_NOTIFYCALLBACK

Pointer to a function used for notification callbacks.

```
typedef (*PFNMIDI_NOTIFYCALLBACK)
        (ULONG ulNotification, ULONG ulParam);
```

PFNVSDENTRY

Pointer to a [VSDEntry](#) routine.

```
typedef FNVSDENTRY *PFNVSDENTRY;
```

In the header file, this is a two-part definition as shown below:

```
typedef ULONG (APIENTRY FNVSDENTRY) (HVSD hvsd, ULONG ulFunc,
                                     ULONG ulFlags, PVOID pParms);
typedef FNVSDENTRY * FPNVSDENTRY;
```

PFNWP

Pointer to a window procedure.

This is the standard function definition for window procedures.

```
typedef FPNWP *PFNWP;
```

The first argument ([HWND](#)) is the handle of the window receiving the message. The second argument ([ULONG](#)) is a message identifier. The third argument ([MPARAM](#)) is the first message parameter (mp1). The fourth argument ([MPARAM](#)) is the second message parameter (mp2). The function returns an [HRESULT](#). Each message has a specific set of possible return codes. The window procedure must return a value that is appropriate for the message being processed.

In the header file, this is a two-part definition as shown below:

```
typedef HRESULT (EXPENTRY FNWP)(HWND, ULONG, MPARAM, MPARAM);
typedef FNWP *PFNWP;
```

Window procedures must be EXPORTED in the definitions file used by the linker.

PLAYL_EVCB

This structure describes a playlist cue point or message event. These events are used only when streaming data to or from the memory stream handler. Playlist events are reported using this data structure. The application media control device must support notification of these events. These events are enabled implicitly at stream creation and cannot be disabled.

```
typedef struct _PLAYL_EVCB {
    ULONG      ulType;           /* Event type. */
    ULONG      ulSubType;        /* Event subtype. */
    ULONG      ulFlags;          /* Flags. */
    HSTREAM     hstream;         /* Stream handle. */
    HID         hid;             /* Stream handler ID. */
    ULONG      ulStatus;         /* Status. */
    ULONG      ulMessageParm;    /* Message parameter. */
    ULONG      unused1;          /* Not used. */
    ULONG      unused2;          /* Not used. */
} PLAYL_EVCB;

typedef PLAYL_EVCB *PPLAYL_EVCB;
```

PLAYL_EVCB Field - ulType

ulType ([ULONG](#))

Identifies the event type. For this event, the type must be `EVENT_IMPLICIT_TYPE`.

PLAYL_EVCB Field - ulSubType

ulSubType ([ULONG](#))

Identifies the event subtype. This field is used to further subdivide an event type. Following is a list of system-defined implicit events:

EVENT_PLAYLISTCUEPOINT

Playlist cuepoint event. This event is reported by the target stream handler when the memory stream handler is the source for a stream instance. The memory stream handler passes the cuepoint information attached to a stream buffer when calling the Sync/Stream Manager. The Sync/Stream Manager passes the cuepoint information to the target stream handler attached to the same buffer.

EVENT_PLAYLISTMESSAGE

Playlist message event. This event is reported by the memory stream handler when a playlist `MESSAGE_OPERATION` instruction is interpreted by the memory stream handler.

PLAYL_EVCB Field - ulFlags

ulFlags (ULONG)

Set of bit flags with various meanings.

PLAYL_EVCB Field - hstream

hstream (HSTREAM)

Handle that identifies the stream instance for this event. This field must be filled in by the application media control device for the [SpiEnableEvent](#) call when this event is enabled. It also must be filled in by the stream handler when reporting this event. index.

PLAYL_EVCB Field - hid

hid (HID)

ID that identifies the stream handler that reported this event. This field can be filled in by the application media control device for the [SpiEnableEvent](#) call when this event is enabled. In this case, it identifies which stream handler must report this event. The stream handler must be able to report this event for the [SpiEnableEvent](#) call to succeed. This field also must be filled in by the stream handler when reporting this event.

PLAYL_EVCB Field - ulStatus

ulStatus (ULONG)

Contains the playlist instruction number from which this event originated.

PLAYL_EVCB Field - ulMessageParm

ulMessageParm (ULONG)

Contains a message parameter that corresponds to the playlist MESSAGE_OPERATION instruction *operand 2* field.

PLAYL_EVCB Field - unused1

unused1 (ULONG)
Not used.

PLAYL_EVCB Field - unused2

unused2 (ULONG)
Not used.

PMIXEREVENT

Pointer to a mixer-event callback function.

```
typedef MIXEREVENT *PMIXEREVENT;
```

In the header file, this is a two-part definition as shown below:

```
typedef LONG (APIENTRY MIXEREVENT)  
            (ULONG          ulStatus,  
             PMCI_MIX_BUFFER pBuffer,  
             ULONG          ulFlags);
```

```
typedef MIXEREVENT *PMIXEREVENT;
```

Valid values for *ulFlags* are:

- MIX_STREAM_ERROR
- MIX_READ_COMPLETE
- MIX_WRITE_COMPLETE

Valid values for *ulStatus* are:

- ERROR_DEVICE_UNDERRUN
- ERROR_DEVICE_OVERRUN

Note:

For *ulStatus* to be set, the value of *ulFlags* must be MIX_STREAM_ERROR.

MIX_STREAM_ERROR will not be set without a previous MIX_READ_COMPLETE or MIX_WRITE_COMPLETE.

The event handler will not be called unless a value has been returned for *ulFlags*.

PMIXERPROC

Pointer to a mixer read or write routine. This routine allows an application to write or read buffers to or from the mixer.

```
typedef MIXERPROC *PMIXERPROC;
```

In the header file, this is a two-part definition as shown below.

```
typedef LONG (APIENTRY MIXERPROC)
    (ULONG          ulHandle,
     PMCI_MIX_BUFFER pBuffer,
     ULONG          ulFlags);

typedef MIXERPROC *PMIXERPROC;
```

The following errors can be returned by the MIXERPROC function:

MCIERR_INVALID_MODE
Mixer mode does not match request.

MCIERR_INVALID_BUFFER
Caller sent an invalid buffer.

MCIERR_INVALID_MODE is returned if the application specified playback mode for the mixer and attempted to read data or if the application specified record mode for the mixer and attempted to write data.

PMMIOPROC

Pointer to an I/O procedure entry point.

```
typedef MMIOPROC FAR *PMMIOPROC;
```

All I/O procedure messages are passed to an I/O procedure using this interface. The syntax for a direct I/O procedure call is shown below.

```
typedef LONG (APIENTRY MMIOPROC)
    (PVOID pmmioinfo,
     USHORT usMsg,
     LONG lParam1,
     LONG lParam2);
```

PSHCFN

Pointer to a stream handler command (SHC) entry routine.

```
typedef SHCFN (FAR *PSHCFN);
```

In the header file, this is a two-part definition as show below:

```
typedef ULONG (APIENTRY SHCFN) (PVOID pParmIn);
typedef SHCFN (FAR *PSHCFN);
```

PSHDFN

Pointer to a function prototype for a stream handler device entry routine.

```
typedef ULONG (FAR *PSHDFN) (PVOID pParmIn);
```

In the header file, this is a two-part definition as shown below:

```
typedef ULONG (FAR *PSHDFN) (PVOID pParmIn);
typedef PVOID PSHDFN;
```

PSMHFN

Pointer to a Sync/Stream Manager Helper (SMH) entry routine.

```
typedef SMHFN (FAR *PSMHFN);
```

In the header file, this is a two-part definition as shown below:

```
typedef ULONG (APIENTRY SMHFN) (PVOID pParmIn);
typedef SMHFN (FAR *PSMHFN);
```

PSZ

Pointer to a null-terminated string.

If you are using C++ **, you may need to use PCSZ.

```
typedef unsigned char *PSZ;
```

QOS

This data structure defines generic parameters for the [QOSInfo](#) structure. It is used on an [MMIOM_BEGINSTREAM](#) message call to an I/O procedure.

```
typedef struct _QOS {
    LONG    IQOSParmId; /* QOS parameter name. */
    LONG    IQOSValue; /* Value of the QOS parameter. */
} QOS;

typedef QOS *PQOS;
```

QOS Field - IQOSParmId

IQOSParmId ([LONG](#))

Quality of service parameter ID can be one of the following:

SERVICE_REQUEST

Requests the type of service required for this stream. The */QOSValue* field contains the type of service.

MAX_EE_JITTER

The number of stream buffers for handling jitter. Buffers needed to store a single unit of data are separate. The */QOSValue* field contains the number of buffers.

MAX_DATA_RATE

Maximum data rate in bytes per second. The */QOSValue* field contains this information.

AVG_DATA_RATE

Average data rate in bytes per second. The */QOSValue* field contains this information.

QOS Field - IQOSValue

IQOSValue (LONG)

The value of the quality of service parameter ID specified in the */QOSParamId* field. If the */QOSParamId* field contains SERVICE_REQUEST, then this field can be one of the following flags:

GUARANTEED

The application requests guaranteed service and if requested QOS is unavailable, the connection is not made.

DONTCARE

The applications requests the given QOS, but if it is unavailable, a connection may be made without quality of service reservation.

DONTRESERVE

The application does not want guarantees (same as no message). The remaining fields are meaningless and not examined.

QOSInfo

This structure defines a guaranteed band width across networks supporting Quality of Service (QOS) and is used on an [MMIOM_BEGINSTREAM](#) message call to an I/O procedure.

```
typedef struct _QOSInfo {  
    LONG    lNumQOSParms; /* Number of QOS structures. */  
    QOS     QOSParms[1]; /* Array of the QOS structures. */  
} QOSInfo;
```

```
typedef QOSInfo *PQOSInfo;
```

QOSInfo Field - INumQOSParms

INumQOSParms (LONG)

Number of [QOS](#) structures in the *QOSParms* field.

QOSInfo Field - QOSParms[1]

QOSParms[1] ([QOS](#))
Array of the [QOS](#) structures.

RECORDTAB

This structure represents a single record of a buffer on a [MMIOM_MULTITRACKREAD](#) message call to an I/O procedure. An array of these structures is passed to the I/O procedure on a multitrack read request. The I/O procedure will use each entry in the array to point to an individual record of data read from a track of a movie file. The *pRecordTabList* field of the [TRACKMAP](#) structure points to this structure. This structure is another representation of the [SRCBUFTAB](#) structure.

```
typedef struct _RECORDTAB {
    ULONG      ulReserved1; /* Reserved. */
    PVOID      pRecord;     /* Points to record in buffer. */
    ULONG      ulLength;    /* Length of record. */
    ULONG      ulReserved2; /* Reserved for system use. */
    ULONG      ulReserved3; /* Reserved for system use. */
    ULONG      ulParm1;     /* Record-specific data. */
    ULONG      ulParm2;     /* Record-specific data. */
} RECORDTAB;

typedef RECORDTAB FAR *PRECORDTAB;
```

RECORDTAB Field - ulReserved1

ulReserved1 ([ULONG](#))
Reserved for system use.

RECORDTAB Field - pRecord

pRecord ([PVOID](#))
Pointer to a data record in *pBuffer* data buffer.

RECORDTAB Field - ulLength

ulLength ([ULONG](#))
Length of the record in bytes.

RECORDTAB Field - ulReserved2

ulReserved2 ([ULONG](#))
Reserved for system use.

RECORDTAB Field - ulReserved3

ulReserved3 ([ULONG](#))
Reserved for system use.

RECORDTAB Field - ulParm1

ulParm1 ([ULONG](#))
Data record specific data. This field can return one of the following flags on an [MMIOM_MULTITRACKREAD](#) message:

- MMIO_IS_KEY_FRAME**
The data record contains a key or reference frame.
- MMIO_IS_PALETTE**
The data record contains a video palette. This data record is not considered a video frame.
- MMIO_INVISIBLE_FRAME**
The data record contains a video frame that should not be displayed. This flag is part of the seek support for movies. It allows a movie to be seeked to a non-reference frame. This frame needs to be decoded, but should not be displayed.
- MMIO_NULL_FRAME**
The data record contains a video frame of zero length. Some file formats contain these zero length video frames, or null frames, as filler for frames missed during capture of the movie, especially during real-time capture.

RECORDTAB Field - ulParm2

ulParm2 ([ULONG](#))
Data record specific data. This field is used to return the video frame number for this video frame if this track is a video track.

RECORDTABWRITE

This structure represents a single record of a data on an [MMIOM_MULTITRACKWRITE](#) message call to an I/O procedure. An array of these

structures is passed to the I/O procedure on a multitrack write request. Each entry contains one record pointer to data and the I/O procedure will write each record to a track of a movie file. The *pRecordTabList* field of the [TRACKMAP](#) structure points to this structure. This structure is another representation of the [TGTFUFTAB](#) structure.

```
typedef struct _RECORDTABWRITE {
    PVOID      pRecord;      /* Points to record in buffer. */
    ULONG      ulReserved1;  /* Reserved. */
    ULONG      ulLength;     /* Length of data record. */
    ULONG      ulReserved2;  /* Reserved for system use. */
    ULONG      ulReserved3;  /* Reserved for system use. */
    ULONG      ulParm1;      /* Data record specific data. */
    ULONG      ulParm2;      /* Data record specific data. */
} RECORDTABWRITE;

typedef RECORDTABWRITE *PRECORDTABWRITE;
```

RECORDTABWRITE Field - pRecord

pRecord ([PVOID](#))
Pointer to a data record to write.

RECORDTABWRITE Field - ulReserved1

ulReserved1 ([ULONG](#))
Reserved for system use.

RECORDTABWRITE Field - ulLength

ulLength ([ULONG](#))
Length of the data record in bytes.

RECORDTABWRITE Field - ulReserved2

ulReserved2 ([ULONG](#))
Reserved for system use.

RECORDTABWRITE Field - ulReserved3

ulReserved3 ([ULONG](#))
Reserved for system use.

RECORDTABWRITE Field - ulParm1

ulParm1 ([ULONG](#))
Data record specific data. This field can contain one of the following flags on an [MMIOM_MULTITRACKWRITE](#) message:

MMIO_IS_KEY_FRAME
The data record is a key or reference video frame.

MMIO_IS_PALETTE
The data record contains a video palette. This data record is not considered a video frame.

RECORDTABWRITE Field - ulParm2

ulParm2 ([ULONG](#))
Data record specific data. On an [MMIOM_MULTITRACKWRITE](#) message, this field must contain the number of NULL frames that should be inserted before this frame (this RECORDTABWRITE entry). This is used during real-time capture of video to save the number of missed (NULL) frames so that the video stream will play back at the correct rate.

RECTL

Rectangle structure.

```
typedef struct _RECTL {  
    LONG    xLeft;    /* X-coordinate of left-hand edge of rectangle. */  
    LONG    yBottom;  /* Y-coordinate of bottom edge of rectangle. */  
    LONG    xRight;   /* X-coordinate of right-hand edge of rectangle. */  
    LONG    yTop;     /* Y-coordinate of top edge of rectangle. */  
} RECTL;  
  
typedef RECTL *PRECTL;
```

RECTL Field - xLeft

xLeft ([LONG](#))
X-coordinate of left-hand edge of rectangle.

RECTL Field - yBottom

yBottom ([LONG](#))

Y-coordinate of bottom edge of rectangle.

RECTL Field - xRight

xRight ([LONG](#))

X-coordinate of right-hand edge of rectangle.

RECTL Field - yTop

yTop ([LONG](#))

Y-coordinate of top edge of rectangle.

RGB2

RGB color value.

```
typedef struct _RGB2 {  
    BYTE    bBlue;        /* Blue component of the color definition. */  
    BYTE    bGreen;       /* Green component of the color definition. */  
    BYTE    bRed;         /* Red component of the color definition. */  
    BYTE    fcOptions;    /* Entry options. */  
} RGB2;  
  
typedef RGB2 *PRGB2;
```

RGB2 Field - bBlue

bBlue ([BYTE](#))

Blue component of the color definition.

RGB2 Field - bGreen

bGreen (BYTE)
Green component of the color definition.

RGB2 Field - bRed

bRed (BYTE)
Red component of the color definition.

RGB2 Field - fcOptions

fcOptions (BYTE)
These can be ORed together if required:

PC_RESERVED	The color entry is reserved for animating color with the palette manager.
PC_EXPLICIT	The low-order word of the color table entry designates a physical palette slot. This allows an application to show the actual contents of the device palette as realized for other logical palettes. This does not prevent the color in the slot from being changed for any reason.

SETUP_BLITTER

Blitter setup structure.

```
typedef struct _SETUP_BLITTER {
    ULONG      ulStructLen;      /* Length of structure. */
    BOOL       fInvert;         /* Image is inverted on blit. */
    FOURCC     fccSrcColorFormat; /* Source data format. */
    ULONG      ulSrcWidth;      /* Width in pels. */
    ULONG      ulSrcHeight;     /* Height in pels. */
    ULONG      ulSrcPosX;       /* X-coordinate position of source data. */
    ULONG      ulSrcPosY;       /* Y-coordinate position of source data. */
    ULONG      ulDitherType;     /* Dither type. */
    FOURCC     fccDstColorFormat; /* Destination color format. */
    ULONG      ulDstWidth;      /* Destination width in pels. */
    ULONG      ulDstHeight;     /* Destination height in pels. */
    LONG       lDstPosX;        /* Destination X-position (relative). */
    LONG       lDstPosY;        /* Destination Y-position (relative). */
    LONG       lScreenPosX;     /* Screen X-position for output. */
    LONG       lScreenPosY;     /* Screen Y-position for output. */
    ULONG      ulNumDstRects;    /* Number of visible rectangles. */
    PRECTL     pVisDstRects;    /* Pointer to array of dest rectangles. */
} SETUP_BLITTER;

typedef SETUP_BLITTER *PSETUP_BLITTER;
```

SETUP_BLITTER Field - ulStructLen

ulStructLen ([ULONG](#))

Length of the structure based on number of parameters being set.

SETUP_BLITTER Field - flInvert

flInvert ([BOOL](#))

Enables output image to be inverted, or rotated, by 180 degrees. The image is vertically inverted if bit 0 is set and horizontally inverted if bit 1 is set. Inverting the image in this manner is independent of the global configuration for horizontally or vertically inverting the image and independent of the coordinate system interpretation for rotated image display.

Normally a [BOOL](#) data type is used to contain a TRUE or FALSE value. However, because the [BOOL](#) data type is defined as:

```
typedef unsigned long BOOL;
```

it is not limited to containing only TRUE and FALSE. Therefore, in order to maintain compatibility in this instance, the [BOOL](#) field can contain more than one bit flag, without changing the data type of the *flInvert* field in the data structure.

SETUP_BLITTER Field - fccSrcColorFormat

fccSrcColorFormat ([FOURCC](#))

Color space FOURCC of the input (source) image data on [DiveBlitImage](#).

SETUP_BLITTER Field - ulSrcWidth

ulSrcWidth ([ULONG](#))

Width in pels of the image data source rectangle to be transferred on [DiveBlitImage](#).

SETUP_BLITTER Field - ulSrcHeight

ulSrcHeight ([ULONG](#))

Height in pels of the image data source rectangle to be transferred on [DiveBlitImage](#).

SETUP_BLITTER Field - ulSrcPosX

ulSrcPosX ([ULONG](#))

The X-coordinate position of the image data source rectangle. This is the position of the lower-left corner of the portion of the image data to be transferred relative to the lower-left corner of the source image data buffer.

SETUP_BLITTER Field - ulSrcPosY

ulSrcPosY ([ULONG](#))

The Y-coordinate position of the image data source rectangle. This is the position of the lower-left corner of the portion of the image data to be transferred relative to the lower-left corner of the source image data buffer.

SETUP_BLITTER Field - ulDitherType

ulDitherType ([ULONG](#))

0 is no dither; 1 is 2 x 2 dither.

SETUP_BLITTER Field - fccDstColorFormat

fccDstColorFormat ([FOURCC](#))

Color space FOURCC of the output (destination) image data on [DiveBlitImage](#). The input image data will be converted to this format when [DiveBlitImage](#) is called. Use FOURCC_SCRN to convert to display buffer format when blitting to the screen.

SETUP_BLITTER Field - ulDstWidth

ulDstWidth ([ULONG](#))

Width of destination image in pels.

SETUP_BLITTER Field - ulDstHeight

ulDstHeight ([ULONG](#))

Height of destination image in pels.

SETUP_BLITTER Field - IDstPosX

IDstPosX (LONG)

The X-coordinate position of the rectangle in the destination output buffer into which the output image is to be placed by [DiveBlitImage](#). This is the position of the lower-left corner of the output image rectangle relative to the lower-left corner of the output image buffer or, if specified, relative to the *IScreenPosX*, *IScreenPosY* position.

SETUP_BLITTER Field - IDstPosY

IDstPosY (LONG)

The Y-coordinate position of the rectangle in the destination output buffer into which the output image is to be placed by [DiveBlitImage](#). This is the position of the lower-left corner of the output image rectangle relative to the lower left corner of the output image buffer or, if specified, relative to the *IScreenPosX*, *IScreenPosY* position.

SETUP_BLITTER Field - IScreenPosX

IScreenPosX (LONG)

The X-coordinate screen position used for determining the position of the output image destination rectangle. This position is relative to the lower left corner of the screen. These values should be set to zero if the destination of [DiveBlitImage](#) is not the screen.

SETUP_BLITTER Field - IScreenPosY

IScreenPosY (LONG)

The Y-coordinate screen position used for determining the position of the output image destination rectangle. This position is relative to the lower left corner of the screen. These values should be set to zero if the destination of [DiveBlitImage](#) is not the screen.

SETUP_BLITTER Field - ulNumDstRects

ulNumDstRects (ULONG)

Number of visible rectangles in the array of rectangles pointed to by *pVisDstRects*.

SETUP_BLITTER Field - pVisDstRects

pVisDstRects ([PRECTL](#))

Array of visible rectangles which defines what portions of the source image are to be displayed. The rectangles specified are relative to the lower-left corner of the output buffer or, if specified, relative to the *iScreenPosX*, *iScreenPosY* position.

SETUP_PARM

This data structure contains fields for the *pSetupParm* and *ulSetupParmSize* fields of the [DDCMDSETUP](#) data structure.

```
typedef struct _SETUP_PARM {
    ULONG    ulStreamTime; /* Stream time in milliseconds. */
    ULONG    ulFlags;      /* Flags (input/output). */
} SETUP_PARM;
```

SETUP_PARM Field - ulStreamTime

ulStreamTime ([ULONG](#))

Specifies the stream time in milliseconds.

SETUP_PARM Field - ulFlags

ulFlags ([ULONG](#))

Device drivers should ignore this field. Specifies the following flag:

SETUP_RECURRING_EVENTS

The device driver sets this flag on return from the [DDCMD_SETUP](#) message if the device driver assumes events are recurring events. In this case, the stream handler will not have to re-enable a recurring event each time the event occurs by sending a [DDCMD_CONTROL](#) message to the device driver. This is useful when CUE_TIME or DATA_CUE events are expected to be used as recurring.

SHC_COMMON

This structure contains fields common to all SHC data structures.

```
typedef struct _SHC_COMMON {
    ULONG    ulFunction; /* SMH command function. */
    HID      hid;        /* Handler ID. */
} SHC_COMMON;
```

```
typedef SHC_COMMON *PSHC_COMMON;
```

SHC_COMMON Field - ulFunction

ulFunction (ULONG)
SMH command function.

SHC_COMMON Field - hid

hid (HID)
Handler ID.

SHD_COMMON

This data structure contains common fields between all SHD data structures.

```
typedef struct _shd_common_parm {
    ULONG      ulFunction; /* Function requested by PDD. */
    HSTREAM    hStream;    /* Stream handle. */
} SHD_COMMON;

typedef SHD_COMMON *PSHD_COMMON;
```

SHD_COMMON Field - ulFunction

ulFunction (ULONG)
Specifies the function requested by the PDD.

SHD_COMMON Field - hStream

hStream (HSTREAM)
Specifies the stream handle needed at interrupt time.

SHD_REPORTEVENT

This data structure contains fields for the SHD_REPORT_EVENT message.

```
typedef struct _shd_reportevent_parm {
    ULONG      ulFunction;    /* Function requested by PDD. */
    HSTREAM    hStream;       /* Stream handle. */
    HEVENT     hEvent;        /* Event handle. */
    ULONG      ulStreamTime;  /* Time in milliseconds of stream position. */
} SHD_REPORTEVENT;

typedef SHD_REPORTEVENT *PSHD_REPORTEVENT;
```

SHD_REPORTEVENT Field - ulFunction

ulFunction (ULONG)
Function requested by PDD.

SHD_REPORTEVENT Field - hStream

hStream (HSTREAM)
Stream handle.

SHD_REPORTEVENT Field - hEvent

hEvent (HEVENT)
Event handle passed back to stream handlers.

SHD_REPORTEVENT Field - ulStreamTime

ulStreamTime (ULONG)
Time in milliseconds of stream position.

SHD_REPORTINT

This data structure contains fields for the [SHD_REPORT_INT](#) message.

```
typedef struct _shd_reportint_parm {
    ULONG      ulFunction;    /* Function requested by PDD. */
    HSTREAM    hStream;       /* Stream handle. */
}
```



```

PVOID      pBuffer;          /* Return pointer to last used buffer. */
ULONG      ulFlag;           /* Reason for interrupt. */
ULONG      ulStatus;         /* Return code or bytes read/written. */
ULONG      ulStreamTime;     /* Time in milliseconds of stream position. */
} SHD_REPORTINT;

typedef SHD_REPORTINT *PSHD_REPORTINT;

```

SHD_REPORTINT Field - ulFunction

ulFunction (ULONG)

Specifies the function requested by the PDD.

SHD_REPORTINT Field - hStream

hStream (HSTREAM)

Specifies the stream handle so the stream handler knows which stream to process.

SHD_REPORTINT Field - pBuffer

pBuffer (PVOID)

Returns the pointer to the last used buffer.

SHD_REPORTINT Field - ulFlag

ulFlag (ULONG)

Specifies the reason for the interrupt. This field defines the following flags:

- ERROR
- STREAM_STOP_NOW
- SHD_READ_COMPLETE
- SHD_WRITE_COMPLETE

Note: Do not set SHD_READ_COMPLETE or SHD_WRITE_COMPLETE if not returning a buffer.

The STREAM_STOP_NOW flag is required. It forces the stream handler to release all buffers that have not been returned. Issue this flag if DDCCMD_STOP or if the VSD has remaining buffers.

SHD_REPORTINT Field - ulStatus

ulStatus ([ULONG](#))
Specifies the status for the return code or the bytes read or written.

SHD_REPORTINT Field - ulStreamTime

ulStreamTime ([ULONG](#))
Specifies the time in milliseconds of stream position.

SHORT

Signed integer in the range -32 768 through 32 767.

```
#define SHORT short
```

SLAVE

This structure is used in the [SpiEnableSync](#) call to identify the slave streams for the synchronization group. It contains an array of slave stream handles and start times for each slave.

```
typedef struct _SLAVE {  
    HSTREAM    hstreamSlave; /* Stream handle for slave stream. */  
    MMTIME    mmtimeStart; /* Start time. */  
} SLAVE;  
  
typedef SLAVE FAR *PSLAVE;
```

SLAVE Field - hstreamSlave

hstreamSlave ([HSTREAM](#))
Contains the stream handle for a slave stream.

SLAVE Field - mmtimeStart

mmtimeStart ([MMTIME](#))

Contains the start time of the slave stream relative to the master stream. It is used in the calculation that determines whether a slave is out of sync with the master stream. A value in this field does not change the starting position of the slave stream; [SpiSeekStream](#) must be called to do this.

SMBD

This structure defines the button style, text, and ID for each button to be included in a secondary message box.

```
typedef struct _SMBD {  
    CHAR    achText[MAX_SMBDTEXT + 1]; /* Text of the button. */  
    ULONG   idButton;                  /* Button ID. */  
    LONG    flStyle;                   /* Button style. */  
} SMBD;  
  
typedef SMBD *PSMBD;
```

SMBD Field - achText[MAX_SMBDTEXT + 1]

achText[MAX_SMBDTEXT + 1] ([CHAR](#))
Text of the button.

SMBD Field - idButton

idButton ([ULONG](#))
Button ID returned when user chooses.

SMBD Field - flStyle

flStyle ([LONG](#))
Button style.

SMBINFO

This structure defines the icon, number of buttons, and window style of the secondary message box window.

```
typedef struct _SMBINFO {  
    HPOINTER hIcon; /* Icon handle. */  
    ULONG    cButtons; /* Number of buttons. */  
}
```

```
    ULONG         flStyle;      /* Message box style. */
    HWND          hwndNotify;   /* Reserved. */
    PSMBD         psmbd;        /* Button definitions. */
} SMBINFO;

typedef SMBINFO *PSMBINFO;
```

SMBINFO Field - hIcon

hIcon ([HPOINTER](#))
Icon handle.

SMBINFO Field - cButtons

cButtons ([ULONG](#))
Number of buttons.

SMBINFO Field - flStyle

flStyle ([ULONG](#))
Message box style:

MB_INFORMATION	Information Icon.
MB_QUERY	Question Icon.
MB_WARNING	Warning Icon.
MB_ERROR	Error Icon.
MB_ICONCUSTOM	Custom Icon.

SMBINFO Field - hwndNotify

hwndNotify ([HWND](#))
Reserved.

SMBINFO Field - psmbd

psmbd ([PSMBD](#))
Array of button definitions.

SMH_COMMON

This structure contains fields common to all SMH data structures.

```
typedef struct _SMH_COMMON {  
    ULONG      ulFunction; /* SMH command function. */  
    HID        hid;       /* Handler ID. */  
} SMH_COMMON;  
  
typedef SMH_COMMON FAR *PSMH_COMMON;
```

SMH_COMMON Field - ulFunction

ulFunction ([ULONG](#))
SMH command function.

SMH_COMMON Field - hid

hid ([HID](#))
Handler ID.

SPCB

This structure describes a stream protocol of a specific data type.

```
typedef struct _SPCB {  
    ULONG      ulSPCBLen;      /* SPCB length. */  
    SPCBKEY    spcbkey;       /* SPCB key. */  
    ULONG      ulDataFlags;    /* Data type flags. */  
    ULONG      ulNumRec;       /* Maximum number for records/buffers. */  
    ULONG      ulBlockSize;    /* Block size. */  
    ULONG      ulBufSize;      /* Buffer size. */  
    ULONG      ulMinBuf;       /* Minimum number of buffers. */  
    ULONG      ulMaxBuf;       /* Maximum number of buffers. */  
    ULONG      ulSrcStart;     /* Number of empty buffers. */  
};
```

```

ULONG      ulTgtStart;          /* Number of full buffers. */
ULONG      ulBufFlags;          /* Buffer flags. */
ULONG      ulHandFlags;         /* Stream handler flags. */
MMTIME     mmtimeTolerance;     /* Resync tolerance value. */
MMTIME     mmtimeSync;          /* Sync pulse granularity. */
ULONG      ulBytesPerUnit;      /* Bytes per unit. */
MMTIME     mmtimePerUnit;       /* MMTIME per unit. */
} SPCB;

typedef SPCB FAR *PSPCB;

```

SPCB Field - ulSPCBLen

ulSPCBLen (ULONG)

Specifies the length of the stream protocol control block.

SPCB Field - spcbkey

spcbkey (SPCBKEY)

Data stream type and internal key. The internal key is used to differentiate between multiple SPCBs of the same data stream type. User defines should be addressed to 8,000,000(h) and higher.

SPCB Field - ulDataFlags

ulDataFlags (ULONG)

Attributes of the data type (that is, it specifies whether data or time cue points are supported by this data type).

SPCBDATA_CUEDATA

This data type can support data cue-point events.

SPCBDATA_CUETIME

This data type can support time cue-point events.

SPCBDATA_NOSEEK

Seeking cannot be done on this data type.

SPCBDATA_YIELDTIME

This flag indicates that the *ulBytesPerUnit* field of the SPCB is interpreted as a millisecond yield time value. This yield takes place in between each I/O read operation. The yield time will also dynamically lower during data streaming depending on how many buffers in the stream are full. This flag is mutually exclusive with the SPCBHAND_GENTIME flag and it can only be used with the SPCBBUF_INTERLEAVED flag. The maximum yield time is 1 second.

SPCB Field - ulNumRec

ulNumRec (ULONG)

Maximum number of records per buffer. (This is valid only for split streams).

SPCB Field - ulBlockSize

ulBlockSize (ULONG)

This field can be interpreted in two ways. If the SPCBBUF_INTERLEAVED flag is set, then this field is interpreted as the size of the I/O read operation. This is used to request a read buffer of a smaller size than the *ulBufSize* field. The source stream handler will take the buffer of *ulBufSize* and break up the actual I/O reads into *ulBlockSize* chunks. If *ulBlockSize* is 1, then the *ulBufSize* is used as the default I/O read buffer size. Otherwise, *ulBlockSize* will be the I/O read buffer size. The *ulBufSize* value must be a multiple of the *ulBlockSize* value.

The second interpretation of this field is for non-interleaved streams, when the SPCBBUF_INTERLEAVED is not set. The *ulBlockSize* field represents the atom size of a data item. For example, for digital audio data type PCM 16-bit stereo at a 44.1KB sampling rate, the block size is 4 bytes.

SPCB Field - ulBufSize

ulBufSize (ULONG)

Size of buffer to be used while streaming. Maximum buffer size is 64KB.

SPCB Field - ulMinBuf

ulMinBuf (ULONG)

Minimum number of buffers needed to maintain a constant data stream.

SPCB Field - ulMaxBuf

ulMaxBuf (ULONG)

Maximum (ideal) number of buffers needed. For normal streams, this means the number of buffers that are allocated for the stream. For user-provided buffer streaming, this means the number of buffers that the Sync/Stream Manager can queue for a consumer. This can be used by a source stream handler that gives the same set of buffers to the Sync/Stream Manager repeatedly. If the number of buffers is set to the number of buffers in the set -1, the source stream handler will be able to detect when the target has consumed a buffer so it can be reused. It is assumed that the set of buffers is an ordered set and each buffer is used in the same order each time.

SPCB Field - ulSrcStart

ulSrcStart (ULONG)

Number of *empty* buffers required to start the source stream handler. The value must be at least as big as the maximum number of buffers that would be requested by the source stream handler.

SPCB Field - ulTgtStart

ulTgtStart (ULONG)

Number of *full* buffers required to start the target stream handler. The value must be at least as big as the maximum number of buffers that would be requested by the target stream handler. Usually, a target requires at least two buffers at the start of the stream.

SPCB Field - ulBufFlags

ulBufFlags (ULONG)

Buffer attributes (that is, the user provides buffers, fixed block size, interleaved data type, maximum buffer size).

SPCBBUF_16MEG

The stream buffers can be allocated above the 16MB line. This is used by stream handler device drivers that can support greater than 16MB addresses.

SPCBBUF_FIXEDBUF

The buffer size for this stream handler must be a particular fixed size (for this data type). The SPCBBUF_INTERLEAVED flag (split stream) implies SPCBBUF_FIXEDBUF.

SPCBBUF_INTERLEAVED

This is a split stream. It consists of one input stream of interleaved data that is split into multiple streams of individual data types. Only the source stream handler can set this flag. This flag is mutually exclusive with the SPCBBUF_USERPROVIDED flag. SPCBBUF_FIXEDBUF cannot be used with this flag set.

SPCBBUF_MAXSIZE

The *ulBufSize* field contains the maximum size buffer that this stream handler can handle.

SPCBBUF_NONCONTIGUOUS

Each data buffer is allocated contiguously in physical memory unless both stream handlers set the non-contiguous flag (SPCBBUF_NONCONTIGUOUS). This flag provides the system flexibility in allocating memory. A device driver stream handler might require contiguous memory, while a DLL stream handler might not.

SPCBBUF_USERPROVIDED

The user provides buffers for streaming. The Sync/Stream Manager does not allocate buffers but attempts to lock down user-provided buffers or copy the data to locked buffers. Using this flag affects the performance of streaming. Only a source stream handler can set this flag. This flag is mutually exclusive with the SPCBBUF_INTERLEAVED flag.

SPCB Field - ulHandFlags

ulHandFlags (ULONG)

Stream handler flags (that is, handlers can generate/receive sync pulses, use the Sync/Stream Manager timer as master, and use the non-streaming handler as a NULL handler).

SPCBHAND_GENSYNC

This stream handler can generate sync pulses.

SPCBHAND_GENTIME

This stream handler can keep track of the real stream time. This handler also supports the [SpiGetTime](#) function and cue point events.

SPCBHAND_NOPREROLL

This stream handler cannot preroll its device (that is, for recording streams, the source stream handler cannot be prerolled). If asked to preroll this stream, the Sync/Stream Manager treats this stream as if it were prerolled.

SPCBHAND_NONSTREAM

This stream handler is a non-streaming handler (that is, it is a stream handler that can participate in synchronization but does not stream).

SPCBHAND_NOSYNC

This stream handler can be in a sync group but does not receive or generate sync pulses. (It is useful to group streams so that they can be manipulated as a group).

SPCBHAND_PHYS_SEEK

This stream handler does a seek to a physical device. Other stream handlers adjust stream time only on a seek request. The Sync/Stream Manager always calls this stream handler first in an [SpiSeekStream](#) call.

SPCBHAND_RCVSYNC

This stream handler can receive sync pulses.

SPCBHAND_TIMER

This stream handler or the device (driver) it communicates with cannot support generation of sync pulses on a granularity necessary for synchronization. Therefore, use the stream manager sync timer as the master timer in a sync group.

SPCB Field - mmtimeTolerance

mmtimeTolerance ([MMTIME](#))

It is used to determine whether to send a sync pulse to a specific slave stream handler.

SPCB Field - mmtimeSync

mmtimeSync ([MMTIME](#))

Time interval in *Chinatown units* between sync pulses. Used to save sync pulse generation granularity if this stream handler is a master but cannot generate its own sync pulse.

SPCB Field - ulBytesPerUnit

ulBytesPerUnit ([ULONG](#))

Bytes per unit of time. This is used to do a seek on linear data that is not compressed or of variable length. Also used for [SHC_GET_TIME](#) queries in a stream handler.

SPCB Field - mmtimePerUnit

mmtimePerUnit ([MMTIME](#))

The amount of MMTIME each unit represents. This also is used for the SHC_SEEK and SHC_GETTIME messages.

SPCBKEY

This structure identifies a stream protocol.

```
typedef struct _SPCBKEY {  
    ULONG      ulDataType;      /* Data type for streaming operation. */  
    ULONG      ulDataSubType;    /* Data subtype. */  
    ULONG      ulIntKey;        /* Internal key. */  
} SPCBKEY;  
  
typedef SPCBKEY FAR *PSPCBKEY;
```

SPCBKEY Field - ulDataType

ulDataType ([ULONG](#))

Data type for streaming operation. Primary descriptor of the data that will be used in the stream. Possible values can be found in OS2MEDEF.H.

SPCBKEY Field - ulDataSubType

ulDataSubType ([ULONG](#))

Secondary descriptor of the data that will be used in the stream.

SPCBKEY Field - ulIntKey

ulIntKey ([ULONG](#))

Contains a unique value used to identify one of several stream protocols of the same data type and subtype.

SRCBUFTAB

This structure is the source buffer table for the [PARM_NOTIFY](#) data structure.

```
typedef struct _SRCBUFTAB {
    PVOID      pBuffer;          /* Pointer to buffer. */
    HID        pRecord;          /* Pointer to record in buffer. */
    ULONG      ulLength;         /* Maximum buffer length on GetEmpty. */
    ULONG      ulMessageParm;    /* Message to passed to application. */
    MMTIME     mmtimeOffset;     /* MMTIME offset from beginning of buffer. */
} SRCBUFTAB;

typedef SRCBUFTAB FAR *PSRCBUFTAB;
```

SRCBUFTAB Field - pBuffer

pBuffer ([PVOID](#))
Pointer to buffer.

SRCBUFTAB Field - pRecord

pRecord ([HID](#))
This is for split streams only.

SRCBUFTAB Field - ulLength

ulLength ([ULONG](#))
Maximum buffer length on GetEmpty, Filled (actual) record/buffer length on ReturnFull.

SRCBUFTAB Field - ulMessageParm

ulMessageParm ([ULONG](#))
Message to passed to application.

SRCBUFTAB Field - mmtimeOffset

mmtimeOffset ([MMTIME](#))

MMTIME offset from beginning of buffer.

STATUS_PARM

This data structure contains fields for the *pStatus* and *ulStatusSize* fields of the [DDCMDSTATUS](#) data structure.

```
typedef struct _STATUS_PARM {
    ULONG    ulTime; /* Current position time in milliseconds. */
} STATUS_PARM;
```

STATUS_PARM Field - ulTime

ulTime ([ULONG](#))

Specifies the current position time in milliseconds.

SYNC_EVCB

This structure describes a sync pulse event. This event is used by master stream handlers to report synchronization pulses to the Sync/Stream Manager. Each slave stream handler must also register a SYNC_EVCB with the Sync/Stream Manager that will be used by the Sync/Stream Manager to report synchronization pulse to slave streams. This event cannot be enabled with the [SpiEnableEvent](#) function call.

```
typedef struct _SYNC_EVCB {
    ULONG    ulType; /* Event type. */
    ULONG    ulSubType; /* Not used. */
    ULONG    ulSyncFlags; /* Sync flags. */
    HSTREAM  hstream; /* Stream handle. */
    HID      hid; /* Stream handler ID. */
    ULONG    ulStatus; /* Sync tolerance value for SPCB. */
    MMTIME   mmtimeStart; /* Start time. */
    MMTIME   mmtimeMaster; /* Master stream time. */
    MMTIME   mmtimeSlave; /* Slave stream time. */
} SYNC_EVCB;

typedef SYNC_EVCB *PSYNC_EVCB;
```

SYNC_EVCB Field - ulType

ulType ([ULONG](#))

For this event, the type must be EVENT_SYNC. This event is used by stream handlers only.

SYNC_EVCB Field - ulSubType

ulSubType (ULONG)
Not used.

SYNC_EVCB Field - ulSyncFlags

ulSyncFlags (ULONG)
This field is a set of bit flags with various meanings. These flags are set by the Sync/Stream Manager only. These flags can be reset by a slave stream handler.

SYNCOVERRUN	This flag is set by the Sync/Stream Manager if a synchronization pulse is reported by the master stream handler before the slave stream handler has processed the existing synchronization pulse.
SYNCPOLLING	This flag is set by the Sync/Stream Manager to indicate to a slave stream that a synchronization pulse has been reported by the master stream handler. The SYNC_EVCB fields (<i>mmtimeStart</i> and <i>mmtimeMaster</i>) are filled in by the Sync/Stream Manager before this bit flag is set. It is the responsibility of the slave stream handler to reset this flag to indicate that the synchronization pulse has been processed.

SYNC_EVCB Field - hstream

hstream (HSTREAM)
Stream handle that identifies the stream instance for this event. This field is filled in by the stream handler when registering the slave SYNC_EVCB with the Sync/Stream Manager on return from a [SHC_ENABLE_SYNC](#) message.

SYNC_EVCB Field - hid

hid (HID)
ID that identifies the slave stream handler that *owns* this SYNC_EVCB. This field is filled in by the slave stream handler on return from a [SHC_ENABLE_SYNC](#) message.

SYNC_EVCB Field - ulStatus

ulStatus (ULONG)
Returned from slave stream handler on SHC_ENABLE_SYNC message. It represents the new synchronization tolerance value that

overrides the synchronization value in the SPCB.

SYNC_EVCB Field - mmtimeStart

mmtimeStart (MMTIME)

This field contains the slave stream start time. It is set by the Sync/Stream Manager on a [SpiEnableSync](#) function call.

SYNC_EVCB Field - mmtimeMaster

mmtimeMaster (MMTIME)

This field contains the current master stream time at the time that the synchronization pulse is reported to the Sync/Stream Manager. The Sync/Stream Manager and the slave stream handler will use this value to determine the amount of time the slave is out-of-sync with the master. This field must be 0 on return from a [SHC_ENABLE_SYNC](#) message.

SYNC_EVCB Field - mmtimeSlave

mmtimeSlave (MMTIME)

This field contains the current slave stream time. The slave stream handler must update the slave stream time on a regular interval. Preferably, before or after every I/O request. The Sync/Stream Manager uses this value to compare against the master stream time to detect out-of-sync conditions.

SZ

Null-terminated string (also known as a zero-terminated string).

```
typedef CHAR SZ[];
```

TGTBUFTAB

This structure is the target buffer table for the [PARM_NOTIFY](#) data structure.

```
typedef struct _TGTBUFTAB {
    PVOID      pBuffer;      /* Pointer to buffer. */
    ULONG      ulBufId;       /* Buffer ID. */
    ULONG      ulLength;      /* Buffer length. */
    ULONG      ulMessageParm; /* Message to passed to application. */
    MMTIME     mmtimeOffset;  /* MMTIME offset from beginning of buffer. */
} TGTBUFTAB;
```

typedef TGTBUFTAB FAR *PTGTBUFTAB;

TGTBUFTAB Field - pBuffer

pBuffer (PVOID)
Pointer to buffer.

TGTBUFTAB Field - ulBufId

ulBufId (ULONG)
Passed to the stream handler on GetFull, must be passed to SSM on on ReturnEmpty.

TGTBUFTAB Field - ulLength

ulLength (ULONG)
Filled (actual) buffer length on GetFull, unused on ReturnEmpty

TGTBUFTAB Field - ulMessageParm

ulMessageParm (ULONG)
Message to passed to application.

TGTBUFTAB Field - mmtimeOffset

mmtimeOffset (MMTIME)
MMTIME offset from beginning of buffer.

TIME_EVCB

This structure describes a time cue point event. There are two types of time cue point events: one is used to report a cue point, while the other is used to report an event, as well as to pause the stream. These events can be enabled through an [SpiEnableEvent](#) call.

```
typedef struct _TIME_EVCB {
    ULONG      ulType;          /* Event type. */
    ULONG      ulSubType;       /* Not used. */
    ULONG      ulFlags;         /* Flags. */
    HSTREAM     hstream;        /* Stream handle. */
    HID         hid;            /* Stream handler ID. */
    ULONG      ulStatus;        /* Status of event. */
    MMTIME      mmtimeStream;   /* MMTIME stream. */
    ULONG      unused1;         /* Not used. */
    ULONG      unused2;         /* Not used. */
} TIME_EVCB;

typedef TIME_EVCB FAR *PTIME_EVCB;
```

TIME_EVCB Field - ulType

ulType (ULONG)

Identifies the event type. The event must be one of the following:

EVENT_CUE_TIME

A cue point in terms of stream time from the start of the stream. This event can be enabled as a single event or recurring event. *Recurring events* are events that are defined as a time interval. An event is generated on each occurrence of this time interval. Single events remain enabled, even after they are reported. In case the stream is seeked backwards in time to a position before a cue point, and play is resumed, the cue point is reported again.

EVENT_CUE_TIME_PAUSE

A cue point in terms of stream time from the start of the stream. This event causes the stream to be paused when the cue point is reached. This event must be enabled as a single event.

TIME_EVCB Field - ulSubType

ulSubType (ULONG)

Not used.

TIME_EVCB Field - ulFlags

ulFlags (ULONG)

Set of bit flags with various meanings.

EVENT_SINGLE

The event is reported on the first detected occurrence. This event remains enabled until disabled. If a stream seek to a position before the cue point occurrence is performed, the cue point might be detected and reported again. This value is the default.

EVENT_RECURRING

This event is reported for each occurrence of the data in the data stream. This event remains enabled until

disabled.

TIME_EVCB Field - hstream

hstream ([HSTREAM](#))

Stream handle that identifies the stream instance for this event. This field must be filled in by the application media control device for the SpiEnableEvent call when this event is enabled. It also must be filled in by the stream handler when reporting this event.

TIME_EVCB Field - hid

hid ([HID](#))

ID that identifies the stream handler that reported this event. This field can be filled in by the application media control device for the SpiEnableEvent call when this event is enabled. In this case, it identifies which stream handler must report this event. The stream handler must be able to report this event for the SpiEnableEvent call to succeed. This field also must be filled in the stream handler when reporting this event.

TIME_EVCB Field - ulStatus

ulStatus ([ULONG](#))

Status of event.

TIME_EVCB Field - mmtimeStream

mmtimeStream ([MMTIME](#))

Stream time at which the data cue point was detected.

TIME_EVCB Field - unused1

unused1 ([ULONG](#))

Not used.

TIME_EVCB Field - unused2

unused2 (ULONG)
Not used.

TRACKMAP

This structure contains information used on [MMIOM_MULTITRACKREAD](#) and [MMIOM_MULTITRACKWRITE](#) message calls to an I/O procedure. It defines track-specific information for one track. The *pTrackMapList* field of the [MMMULTITRACKREAD](#) and [MMMULTITRACKREAD](#) structures point to this structure.

```
typedef struct _TRACKMAP {
    ULONG      ulTrackID;          /* Track identifier. */
    ULONG      ulNumEntries;       /* Number of record entries. */
    PRECORDTAB pRecordTabList;    /* Pointer to a record table. */
} TRACKMAP;

typedef TRACKMAP *PTRACKMAP;
```

TRACKMAP Field - ulTrackID

ulTrackID (ULONG)
Track identifier.

TRACKMAP Field - ulNumEntries

ulNumEntries (ULONG)
The number of entries in *pRecordTabList*. On output it returns the number.

TRACKMAP Field - pRecordTabList

pRecordTabList (PRECORDTAB)
Pointer to a record table.

ULONG

32-bit unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long ULONG;
```

USHORT

Unsigned integer in the range 0 through 65 535.

```
typedef unsigned short USHORT;
```

VOID

A data area of undefined format.

```
#define VOID void
```

VSD_DCB

This structure contains device specific-information. It is used at stream creation to identify the VSD DLL to be used by the Ring 3 audio stream handler.

```
typedef struct _VSD_DCB {
    ULONG      ulDCBLen;           /* Control block length. */
    SZ         szDevName[MAX_SPI_NAME]; /* Device driver name. */
    ULONG      ulSysFileNum;       /* File handle number. */
    ULONG      hvsd;              /* Handle to VSD instance. */
    PFN        pfnvsdEntryPoint;   /* Pointer to address of VSD entry point. */
    ULONG      ulReserved1;        /* Reserved. */
    ULONG      ulReserved2;        /* Reserved. */
} VSD_DCB;

typedef VSD_DCB FAR *PVSD_DCB;
```

VSD_DCB Field - ulDCBLen

ulDCBLen (ULONG)
Length of the device control block.

VSD_DCB Field - szDevName[MAX_SPI_NAME]

szDevName[\[MAX_SPI_NAME\]](#) ([SZ](#))
Identifies the device driver that the stream handler connects to for a particular stream instance. For stream handler device drivers, inter-device driver communication (IDC) is used to call the physical device driver. The device driver name must not be more than eight characters (excluding the file extension).

VSD_DCB Field - ulSysFileNum

ulSysFileNum ([ULONG](#))
File handle number.

VSD_DCB Field - hvsd

hvsd ([ULONG](#))
Handle to VSD instance.

VSD_DCB Field - pfnvsdEntryPoint

pfnvsdEntryPoint ([PFN](#))
Pointer to address of VSD entry point.

VSD_DCB Field - ulReserved1

ulReserved1 ([ULONG](#))
Reserved.

VSD_DCB Field - ulReserved2

ulReserved2 ([ULONG](#))
Reserved.

WAVE_HEADER

This structure is a subheader for the audio header and is usually contained within the [MMXWAV_HEADER](#) structure. It contains descriptive information about the digital audio element, such as sample rate, or bits per sample. It is part of the standard presentation format for audio data and is returned on [mmioGetHeader](#) and used on [mmioSetHeader](#) when the [HMMIO](#) refers to an open audio or to an open movie file when the active track for this file is set to an audio track.

```
typedef struct _WAVE_HEADER {
    USHORT    usFormatTag;        /* Type of wave format. */
    USHORT    usChannels;        /* Number of channels. */
    ULONG     ulSamplesPerSec;    /* Sampling rate. */
    ULONG     ulAvgBytesPerSec;  /* Average bytes per second. */
    USHORT    usBlockAlign;      /* Block alignment in bytes. */
    USHORT    usBitsPerSample;   /* Bits per sample. */
} WAVE_HEADER;

typedef WAVE_HEADER *PWAVE_HEADER;
```

WAVE_HEADER Field - usFormatTag

usFormatTag ([USHORT](#))
A number indicating the wave format category of data. Valid values for this field are as follows:

DATATYPE_WAVEFORM	Waveform audio (PCM)-this is the only valid value for standard presentation format.
DATATYPE_ALAW	A-Law
DATATYPE_MULAW	Mu-Law
DATATYPE_ADPCM_AVC	ADPCM audio

WAVE_HEADER Field - usChannels

usChannels ([USHORT](#))
Number of channels represented in the waveform data; Channel 1 for monaural and Channel 2 for stereo.

CH_1	Mono
CH_2	Stereo
CH_3	Quad

WAVE_HEADER Field - ulSamplesPerSec

ulSamplesPerSec (ULONG)

Sampling rates, in kilohertz, at which each channel should be played. Valid values for standard audio presentation are 11.025, 22.05, and 44.1 kHz.

HZ_8000	8000	/*	8.000 kHz	*/
HZ_11025	11025	/*	11.025 kHz	*/
HZ_14700	14700	/*	14.700 kHz (SPV/2)	*/
HZ_18900	18900	/*	18.900 kHz (CD/XA LVL C)	*/
HZ_22050	22050	/*	22.050 kHz	*/
HZ_37800	37800	/*	37.800 kHz (CD/XA LVL B)	*/
HZ_44100	44100	/*	44.100 kHz	*/

WAVE_HEADER Field - ulAvgBytesPerSec

ulAvgBytesPerSec (ULONG)

Rate at which the waveform data should be transferred. Playback software can estimate the buffer size by using this value. The following formula can be used to calculate this value for the audio presentation format of standard PCM data:

$$\text{usChannels} \times \text{usBitsPerSec} \times \text{usBitsPerSample} / 8$$

WAVE_HEADER Field - usBlockAlign

usBlockAlign (USHORT)

Block alignment, in bytes, of the waveform data. Playback software needs to process a multiple of *usBlockAlign* bytes of data at a time. This value can be used for buffer alignment. It should be determined by the following formula for audio presentation format of standard PCM data:

$$\text{usChannels} \times \text{usBitsPerSample} / 8$$

WAVE_HEADER Field - usBitsPerSample

usBitsPerSample (USHORT)

Number of bits of data used to represent each sample of each channel. If multiple channels are used, the sample size is the same for each channel.

BPS_4	4	/*	4 bits/sample (ADPCM)	*/
BPS_8	8	/*	8 bits/sample (PCM)	*/
BPS_16	16	/*	16 bits/sample (PCM)	*/

XDIBHDR_PREFIX

This structure is a subheader for both the image and video header and is usually contained within the [MMXDIBHEADER](#) structure. It is part of the standard presentation format for image and video data and is returned on [mmioGetHeader](#) and used on [mmioSetHeader](#) when the [HMMIO](#) refers to an open image or open movie file.

```
typedef struct _XDIBHDR_PREFIX {
    ULONG      ulMemSize;      /* Length of bitmap. */
    ULONG      ulPelFormat;    /* FOURCC code defining the pel format. */
    USHORT     usTransType;    /* Transparency color. */
    ULONG      ulTransVal;     /* Transparent color indicator. */
} XDIBHDR_PREFIX;

typedef XDIBHDR_PREFIX *PXDIHDR_PREFIX;
```

XDIBHDR_PREFIX Field - ulMemSize

ulMemSize ([ULONG](#))
Size of the bitmap bits.

XDIBHDR_PREFIX Field - ulPelFormat

ulPelFormat ([ULONG](#))
Specifies the FOURCC code defining the pel format of the bitmap data. Valid values for this field would be as follows for the standard image presentation format:

- 'rgbb' - Pel format is packed, unpalettized, 24-bit RGB.
 - 'palb' - Pel format is 1, 4, or 8-bit palettized data, depending on the value in the *cBitCount* field.
-

XDIBHDR_PREFIX Field - usTransType

usTransType ([USHORT](#))
Transparency color. This field is currently not used in the standard presentation format.

XDIBHDR_PREFIX Field - ulTransVal

ulTransVal ([ULONG](#))

Transparent color indicator. This field is currently not used in the standard presentation format.

XWAV_HEADERINFO

This structure is a subheader of the audio header and is usually contained within the [MMXWAV_HEADER](#) structure. It contains information about the length of the standard presentation format for audio data and is returned on [mmioGetHeader](#) and used on [mmioSetHeader](#) when the [HMMIO](#) refers to an open audio or to an open movie file when the active track for this file is set to an audio track.

```
typedef struct _XWAV_HEADERINFO {
    ULONG        ulAudioLengthInMS;        /* Audio length in milliseconds. */
    ULONG        ulAudioLengthInBytes;     /* Audio length in bytes. */
    PVOID        pAdditionalInformation;    /* Pointer to specific info. */
} XWAV_HEADERINFO;

typedef XWAV_HEADERINFO *PXWAV_HEADERINFO;
```

XWAV_HEADERINFO Field - ulAudioLengthInMS

ulAudioLengthInMS ([ULONG](#))

Total length of the audio data in milliseconds, or 0. Use of this field is optional.

XWAV_HEADERINFO Field - ulAudioLengthInBytes

ulAudioLengthInBytes ([ULONG](#))

Total length of the audio data in bytes, or 0. Use of this field is optional.

XWAV_HEADERINFO Field - pAdditionalInformation

pAdditionalInformation ([PVOID](#))

Pointer to additional information in the extended chunk for non-PCM types.

Types of MIDI Messages

There are three types of MIDI system messages:

- System common
- System exclusive
- System real time

System-common messages are intended for all receivers, regardless of channel. *System-exclusive* messages are defined by the manufacturer and are not limited in length. *System-real-time* messages can interrupt other messages because they carry timing information. For example, a long system message might have an embedded timer message within it. Other MIDI events cannot occur within a system message or nested system message; however, a system message can occur in the middle of other MIDI events.

While in MIDI mode, almost all functions provided by audio device drivers can be controlled by the application by using system-exclusive messages.

General Format of System-Exclusive Messages

The general format of system-exclusive messages is shown in the following table:

Message Description	Hexadecimal Format
Timing compression (long)	F0 00 00 3A 01 <i>lsb</i> <i>msb</i> F7
Timing compression (short)	F0 00 00 3A 07-7F F7
Sound generator commands	F0 00 00 3A 02 <i>unit</i> <i>cmd</i> <i>chan</i> ... F7
Device driver control	F0 00 00 3A 03 <i>cmd</i> ... F7
Device driver query	F0 00 00 3A 04 <i>qid</i> F7
Device driver response	F0 00 00 3A 05 <i>qid</i> <i>data</i> F7
Timbre parameter	F0 00 00 3A 06 <i>cmd</i> <i>hh</i> <i>mm</i> <i>ll</i> ... F7

The following list contains the categories of system exclusive messages:

- [Device Driver Control Messages](#)
- [Device Driver Query Messages](#)
- [Device Driver Response Messages](#)
- [Sound Generator Commands Messages](#)
- [Timbre Parameter Messages](#)
- [Timing Compression Messages](#)

Device Driver Control Messages

The following are types of device driver control messages:

- Balance
- Generic sound selection
- Master volume
- Tempo
- Timing generation
- Volume

Balance Control

This message sets the balance level for the audio device driver:

```
F0 00 00 3A 03 08 b1 d1 dm F7
```

bb = Balance level (0 = full left, 40 = middle, 7F = full right)
dl = Duration LSB
dm = Duration MSB

Balance is specified as a relative value from 0 - 7F, where 0 is full left, 40 is middle, 7F is full right. The duration specifies the length of time (in tenths of a second) necessary to gradually go from the current level to the target level. For example, to set the balance to full right (7F) with duration set to 10 to gradually move the new balance to the right over a one-second period, the balance control message would be:

```
F0 00 00 3A 03 08 7F 0A 00 F7
```

Generic Sound Selection

This message selects the generic sound for an instrument:

```
F0 00 00 3A 03 03 ii tt ss F7
```

ii = Instrument number
tt = Sound type (0 = Musical instrument, 1 = sound effects)
ss = Sound ID

There are 128 generic musical instrument sounds and 128 sound effects types. When a requested sound is not available on a device, the nearest available sound is used. It is recommended that general MIDI mode be used instead of this message.

Master Volume Control

This message sets the master volume level for the audio device driver:

```
F0 00 00 3A 03 09 v1 vm 00 F7
```

v1 = Master Volume LSB
vm = Master Volume MSB

Master volume is intended to permit immediate control of overall volume without interfering with the current volume setting as set by the volume control message, described in **Volume Control** below. *Volume* is specified as a relative value from 0 - 3FFF. For example, to set the volume to maximum (3FFF) the master volume control message would be:

```
F0 00 00 03 03 09 7F 7F 00 F7
```

This message causes a MIDI volume control change message to be generated. Any subsequent volume control change messages encountered are scaled proportionately. Actual volume is the product of volume and master volume.

Tempo Control

This message sets the tempo for system real-time timing clock generation:

```
F0 00 00 3A 03 02 t1 tm dd F7
```

t1 = Tempo LSB
tm = Tempo MSB
dd = Duration

Notice that *tempo* is expressed as 1/10 beats per minute. The duration specifies the length of time (in tenths of a second) necessary to go gradually from the current tempo to the specified tempo. For example, to set the tempo to 120 beats per minute, with duration set to 0 to make the new tempo effective immediately, the tempo control message would be:

```
120 bpm = 1200 1/10 increments
1200 decimal = 00000100 10110000 binary
7 bit MSB (tm) = 000100 1 = hex 9
7 bit LSB (t1) = 0110000 = hex 30

F0 00 00 3A 03 02 30 09 00 F7
```

Timing Generation Control

This message tells the device driver what timing information to generate:

```
F0 00 00 3A 03 01 tt pp 00 F7

tt = System real-time control flags (default 08H)
0x1x xxxx Output timing clocks to MIDI-Out
0x0x xxxx Do not output timing clocks to MIDI-Out (default)
0xx1 xxxx Merge timing clocks with MIDI-In
0xx0 xxxx Do not pass timing clocks to MIDI-In (default)
0xxx 1xxx Synchronize output to timing clocks (default)
0xxx 0xxx Disable output synchronization
0xxx x1xx Perform timing data compression
0xxx x0xx Disable timing data compression (default)
0xxx xxRR Reserved bits - always 0

pp = System real-time 24 CPQN rate prescaler
00dd dddd Multiply by ddddd+1 (1-64) for 24 to 1536 ppq
01dd dddd Divide by ddddd+1*3 (3-192) for 8 to 1/8 ppq
```

By default, device drivers utilize real-time messages at a rate of 24 per quarter note (beat) and 120 beats per minute internally for performing output synchronization. However, they do not queue timing clocks to either MIDI-Out or MIDI-In (Read data).

Volume Control

This message sets the volume level for the audio device driver. Volume is specified as a relative value from 0-7F.

```
F0 00 00 3A 03 07 vv d1 dm F7

vv = Volume (0-7F)
d1 = Duration LSB
dm = Duration MSB
```

The duration specifies the length of time (in tenths of a second) necessary to gradually go from the current level to the target level. For example, to set the volume to maximum (7F) with duration set to 0 to make the new volume effective immediately, the volume control message would be:

```
F0 00 00 3A 03 07 7F 00 00 F7
```

This message causes subsequent MIDI volume control change messages to be generated. Any subsequent volume control change message encountered will be scaled proportionately. Notice that the actual volume is the product of volume and master volume.

Device Driver Query Messages

The following are types of device driver query messages:

- Query device capability
- Query Device ID
- Query output queue size

Query Device Capability

This message is issued to a device driver by an application to query the basic capability of a device driver:

```
F0 00 00 3A 04 01 F7
```

The device driver responds to this message by returning a device capability response message (see [Device Driver Response Messages](#)).

Query Device ID

This message is issued to a device driver by an application to request the identity of the device driver:

```
F0 00 00 3A 04 04 F7
```

The device driver responds to this message by returning a device ID response message, which identifies the device (see [Device Driver Response Messages](#)).

Query Output Queue Size

This message is issued to a device driver by an application to request the size of the output queue:

```
F0 00 00 3A 04 02 F7
```

The device driver responds to this message by returning a queue size response message (see [Device Driver Response Messages](#))

Device Driver Response Messages

The following are types of device driver response messages:

- Device capability
- Device ID
- Queue size

Device Capability Response

This message is sent by a device driver in response to a query device capability message (see [Device Driver Query Messages](#)). It indicates the basic MIDI capability of the device driver.

```
F0 00 00 3A 05 01 ii mm 00 F7
```

ii = Number of sound generators (instruments)

mm = MIDI capability flags

01xx xxxx MIDI input is supported

0x1x xxxx MIDI output is supported

0xx1 xxxx System real-time Timing clocks are supported

Device ID Response

This message identifies the device and is sent by a device driver in response to a query device ID message (see [Device Driver Query Messages](#)).

```
F0 00 00 3A 05 04 b1 b2 b3 F7
```

b1 = 1st byte of Device ID

b2 = 2nd byte of Device ID

b3 = 3rd byte of Device ID

This message is provided to permit applications to perform device unique operations such as issuing device-specific system exclusive messages.

Queue Size Response

This message is sent by a device driver in response to a query output queue size message (see [Device Driver Query Messages](#)). It indicates the total output queue size in bytes.

```
F0 00 00 3A 05 02 l1 mm 00 F7
```

l1 = Lower 7 bits of queue size

mm = Upper 7 bits of queue size

(Queue Size in bytes = 00mm mmmml111 l111)

For example, if the device driver's output queue is 512 bytes in length, the returned data is:

```
F0 00 00 3A 05 02 00 04 00 F7
```

Sound Generator Commands Messages

These messages are specific to a certain device. Although not part of the audio device driver's architecture, they are included here. Notice in the following example that the 6th MIDI byte identifies the specific IBM device that is addressed:

M-ACPA (IBM Audio Capture & Playback Adapter)

Program Change Enable

This message instructs the M-ACPA synthesizer to honor or ignore MIDI program change messages (Cn).

F0 00 00 3A 02 01 01 **ee** 00 F7

ee = Program Changes Enable (0 = disabled, 1 = enabled; default)

By default, program changes are honored. Notice that this message does not affect generic sound selection messages, which are always honored.

Timbre Parameter Messages

Timbre parameter messages are provided to permit interrogating and changing of the parameters that control the timbre of a given voice. Parameters are addressed using three 7-bit bytes (that is, 21 bits), enabling approximately two million parameters to be addressed. In most cases, these parameters are divided into units such as banks, voices, and so forth. Parameter values are passed in two 7-bit bytes, allowing up to 14-bits per parameter.

The following are types of timbre parameter messages:

- Query timbre parameter
- Request timbre block
- Set timbre parameter
- Timbre block
- Timbre parameter response
- Write timbre block

Query Timbre Parameter

This message is issued to a device driver by an application to request the current setting of a timbre parameter:

F0 00 00 3A 06 01 **hh mm ll** F7

hh = High 7-bits parameter #

mm = Middle 7-bits parameter #

ll = Low 7-bits parameter #

The device driver responds to this message by returning a timbre parameter response message (see below).

Request Timbre Block

This message is issued to a device driver by an application to request a block of timbre data.

F0 00 00 3A 06 04 **hh mm** 00 F7

hh = High 7-bits block #

mm = Middle 7-bits block #

The device driver responds to this message by returning a timbre block message (see below).

Set Timbre Parameter

This message is used to set or change a timbre parameter:

```
F0 00 00 3A 06 03 hh mm ll dl dm F7
```

hh = High 7-bits parameter #
mm = Middle 7-bits parameter #
ll = Low 7-bits parameter #
dl = Parameter data LSB (0-127)
dm = Parameter data MSB

Timbre Block

This message is issued by a device driver in response to a request timber block message (see above). It transfers blocks of timbre data from a device driver to an application.

```
F0 00 00 3A 06 05 hh mm ll < len > F7
```

hh = High 7-bits block #
mm = Middle 7-bits block #
ll = Length of 7-bit data
< len > = Variable length (7-bit) data block

Timbre Parameter Response

This message is sent by a device driver in response to a query timbre parameter message (see above). It returns the current value of a timbre parameter.

```
F0 00 00 3A 06 02 hh mm ll dl dm F7
```

hh = High 7-bits parameter #
mm = Middle 7-bits parameter #
ll = Low 7-bits parameter #
dl = Parameter data LSB (0-127)
dm = Parameter data MSB

Write Timbre Block

This message is issued to a device driver by an application to write a block of timbre data:

```
F0 00 00 3A 06 06 hh mm ll < len > F7
```

hh = High 7-bits block #
mm = Middle 7-bits block #
ll = Length of 7-bit data
< len > = Variable length (7-bit) data block

Timing Compression Messages

These messages are used to compress sequential timing clocks (F8) into a single system exclusive message. Up to 16383 (that is, F8) Timing clocks can be compressed into a single 6- or 8-byte message, which can be written to a device driver to cause a delay between output events if output synchronizing is enabled (see [Device Driver Control Messages](#)).

```
F0 00 00 3A 01 ll mm F7 (Long version)
F0 00 00 3A nn F7 (Short version)
```

ll = Number of System Real Time Timing Clocks LSB
mm = Number of System Real Time Timing Clocks MSB
nn = Number of System Real Time Timing Clocks (nn = 7 through 127)

The short version can be used to represent durations of from 7 - 127 clocks. Longer durations can be represented with the long version, where the number of timing clocks that have occurred is calculated by combining the low 7-bits of *mm* with the low 7-bits of *ll* to produce a

single 16-bit value:

$$0mmmm\ mmmmm + 0111\ 1111 = 00mm\ mmmmm\ m111\ 1111$$

Multimedia Specification Overview

The following sections contain specifications for multimedia file formats. [Resource Interchange File Format](#) describes the Resource Interchange File Format (RIFF) tagged file structure. [Multimedia File Formats](#) describes recognized multimedia file formats, most of which are based on the RIFF tagged file structure.

If your application requires a new file format, it is recommended that it be defined using the RIFF tagged file structure described in [Resource Interchange File Format](#).

Notation Conventions

The following table lists some of the notation conventions used to describe RIFF and the multimedia file formats. Further conventions and the notation for documenting RIFF forms are presented later in [Notation for Representing Sample RIFF Files](#).

Notation	Description
<element label>	RIFF file element with the label "element label"
<element label: TYPE>	RIFF file element with data type "TYPE"
[<element label>]	Optional RIFF file element
<element label>...	One or more copies of the specified element
[<element label>]...	Zero or more copies of the specified element

Registering Multimedia Formats

Several multimedia codes and formats require registration to guarantee their uniqueness. These multimedia elements include the following:

- Compression techniques
- RIFF form types, chunk identifiers, and list types
- Compound file usage codes
- Waveform audio format codes.

More information about registration requirements is included with the descriptions of the RIFF tagged file structure. To register multimedia elements, request a *Multimedia Developer Registration Kit* from the following group:

Microsoft Corporation
Multimedia Systems Group
Product Marketing
One Microsoft Way
Redmond, WA 98052-6399

The *Multimedia Developer Registration Kit* also lists currently defined multimedia elements.

Resource Interchange File Format

RIFF (Resource Interchange File Format) is the tagged file structure developed for multimedia resource files. The structure of a RIFF file is similar to the structure of an Electronic Arts Interchange File Format file (EA IFF). RIFF is not actually a file format itself (because it does not represent a specific kind of information), but its name contains the words "interchange file format" in recognition of its roots in IFF.

RIFF has a counterpart, RIFX, that is used to define RIFF file formats that use the Motorola** integer byte-ordering format rather than the Intel** format. A RIFX file is the same as a RIFF file, except that the first four bytes are 'RIFX' instead of 'RIFF', and integer byte ordering is represented in Motorola format.

Chunks

The basic building block of a RIFF file is called a *chunk*. Using C syntax, a chunk can be defined as follows:

```
typedef unsigned long ULONG;
typedef unsigned char BYTE;

typedef ULONG FOURCC;          /* Four-character code */

typedef FOURCC CKID;           /* Four-character-code chunk identifier */
typedef ULONG CKSIZE;          /* 32-bit unsigned size value */

typedef struct {               /* Chunk structure */
    CKID      ckID;             /* Chunk type identifier */
    CKSIZE     ckSize;          /* Chunk size field (size of ckData) */
    BYTE      ckData[ckSize];   /* Chunk data */
} CK;
```

A FOURCC is represented as a sequence of one to four ASCII alphanumeric characters, padded on the right with blank characters (ASCII character value 32) as required, with no embedded blanks.

For example, the four-character code 'FOO' is stored as a sequence of four bytes: 'F', 'O', 'O', " " in ascending addresses. For quick comparisons, a four-character code may also be treated as a 32-bit number.

The three parts of the chunk are described in the following table:

Part	Description
ckID	A four-character code that identifies the representation of the chunk data data . A program reading a RIFF file can skip over any chunk whose chunk ID it doesn't recognize; it simply skips the number of bytes specified by <i>ckSize</i> plus the pad byte, if present.
ckSize	A 32-bit unsigned value identifying the size of <i>ckData</i> . This size value does not include the size of the <i>ckID</i> or <i>ckSize</i> fields or the pad byte at the end of <i>ckData</i> .
ckData	Binary data of fixed or variable size. The start of <i>ckData</i> is word-aligned with respect to the start of the RIFF file. If the chunk size is an odd number of bytes, a pad byte with value zero is written after <i>ckData</i> . Word aligning improves access speed (for chunks resident in memory) and maintains compatibility with EA IFF. The <i>ckSize</i> value does not include the pad byte.

We can represent a chunk with the following notation (in this example, the *ckSize* and pad byte are implicit):

```
<ckID> ( <ckData> )
```

Two types of chunks, the 'LIST' and 'RIFF' chunks, may contain nested chunks, or subchunks. These special chunk types are discussed later in this document. All other chunk types store a single element of binary data in **<ckData>**.

RIFF Forms

A RIFF form is a chunk with a 'RIFF' chunk ID. The term also refers to a file format that follows the RIFF framework. The following is the current list of registered RIFF forms. Each is described in [Multimedia File Formats](#).

Form Type	Description
WAVE	Waveform Audio Format

Using the notation for representing a chunk, a RIFF form looks like the following:

```
RIFF ( <formType> <ck>... )
```

The first four bytes of a RIFF form make up a chunk ID with values 'R', 'I', 'F', 'F'. The *ckSize* field is required, but for simplicity it is omitted from the notation.

The first ULONG of chunk data in the 'RIFF' chunk (shown above as **<formType>**) is a four-character code value identifying the data representation, or *form type*, of the file. Following the form-type code is a series of subchunks. Which subchunks are present depends on the form type. The definition of a particular RIFF form typically includes the following:

- A unique four-character code identifying the form type
- A list of mandatory chunks
- A list of optional chunks
- Possibly, a required order for the chunks

Defining and Registering RIFF Forms

The form-type code for a RIFF form must be unique. To guarantee this uniqueness, you must register any new form types before release. See [Registering Multimedia Formats](#) for information on registering RIFF forms.

Like RIFF forms, RIFX forms must also be registered. Registering a RIFF form does not automatically register the RIFX counterpart. No RIFX form types are currently defined.

Registered Form and Chunk Types

By convention, the form-type code for registered form types contains only digits and uppercase letters. Form-type codes that are all uppercase denote a registered, unique form type. Use lowercase letters for temporary or prototype chunk types.

Certain chunk types are also globally unique and must also be registered before use. These registered chunk types are not specific to a certain form type; they can be used in any form. If a registered chunk type can be used to store your data, you should use the registered chunk type rather than define your own chunk type containing the same type of information.

For example, a chunk with chunk ID 'INAM' always contains the name or title of a file. Also, within all RIFF files, file names or titles are contained within chunks with ID 'INAM' and have a standard data format.

Unregistered (Form-Specific) Chunk Types

Chunk types that are used only in a certain form type use a lowercase chunk ID. A lowercase chunk ID has specific meaning only within the context of a specific form type. After a form designer is allocated a registered form type, the designer can choose lowercase chunk types to use within that form. See [Registering Multimedia Formats](#) for information on registering form types.

For example, a chunk with ID 'scln' inside one form type might contain the "number of scan lines." Inside some other form type, a chunk with ID 'scln' might mean "secondary lambda number."

Notation for Representing Sample RIFF Files

RIFF is a binary format, but it is easier to comprehend as an ASCII representation of a RIFF file. This section defines a standard notation used to present samples of various types of RIFF files. If you define a RIFF form, we urge you to use this notation in any file format samples you provide in your documentation.

Basic Notation for Representing RIFF Files

The following information summarizes the elements of the RIFF notation required for representing sample RIFF files:

Notation	Description														
<ckID> (<ckData>)	<p>The chunk with ID <ckID> and data <ckData>. As previously described, <ckID> is a four-character code which may be enclosed by single quotes for emphasis.</p> <p>For example, the following notation describes a 'RIFF' chunk with a form type of 'QRST'. The data portion of this chunk contains a 'FOO' subchunk.</p> <pre>RIFF('QRST' FOO(17 23))</pre> <p>The following example describes an 'ICOP' chunk containing the string "Copyright Encyclopedia International.":</p> <pre>'ICOP' ("Copyright Encyclopedia International."Z)</pre>														
<number>[<modifier>]	<p>A number in Intel format, where <number> is an optional sign (+ or -) followed by one or more digits and modified by the optional <modifier>. Valid <modifier> values follow:</p> <table><tr><th>Modifier</th><th>Meaning</th></tr><tr><td>None</td><td>16-bit number in decimal format</td></tr><tr><td>H</td><td>16-bit number in hexadecimal format</td></tr><tr><td>C</td><td>8-bit number in decimal format</td></tr><tr><td>CH</td><td>8-bit number in hexadecimal format</td></tr><tr><td>L</td><td>32-bit number in decimal format</td></tr><tr><td>LH</td><td>32-bit number in hexadecimal format</td></tr></table> <p>Several examples follow:</p> <pre>0 65535 -1 0L 4a3c89HL -1C 21HC</pre> <p>Note that -1 and 65535 represent the same value. The</p>	Modifier	Meaning	None	16-bit number in decimal format	H	16-bit number in hexadecimal format	C	8-bit number in decimal format	CH	8-bit number in hexadecimal format	L	32-bit number in decimal format	LH	32-bit number in hexadecimal format
Modifier	Meaning														
None	16-bit number in decimal format														
H	16-bit number in hexadecimal format														
C	8-bit number in decimal format														
CH	8-bit number in hexadecimal format														
L	32-bit number in decimal format														
LH	32-bit number in hexadecimal format														

'<chars>'

application reading this file must know whether to interpret the number as signed or unsigned.

A four-character code (32-bit quantity) consisting of a sequence of zero to four ASCII characters **<chars>** in the given order. If **<chars>** is less than four characters long, it is implicitly padded on the right with blanks. Two single quotes is equivalent to four blanks. Examples follow:

```
'RIFF'  
'xyz '  
' '
```

"<string>"[<modifier>]

<chars> can include escape sequences, which are combinations of characters introduced by a backslash (\) and used to represent other characters. Escape sequences are listed in the following section.

The sequence of ASCII characters contained in **<string>** and modified by the optional modifier **<modifier>**. The quoted text can include any of the escape sequences listed in the following section. Valid **<modifier>** values follow:

Modifier	Meaning
none	No NULL terminator or size prefix.
Z	String is NULL-terminated
B	String has an 8-bit (byte) size prefix
US	String has a 16-bit (ushort) size prefix
BZ	String has a byte-size prefix and is NULL-terminated
WZ	String has a word-size prefix and is NULL-terminated

NULL-terminated means that the string is followed by a character with ASCII value 0. A size prefix is an unsigned integer, stored as a byte or a word in Intel format preceding the string characters, that specifies the length of the string. In the case of strings with BZ or WZ modifiers, the size prefix specifies the size of the string without the terminating NULL.

The various string formats referred to above are discussed in "Storing Strings in RIFF Chunks," following later in this section.,

Examples follow:

```
"No prefix, no NULL terminator"  
"No prefix, NULL terminator"Z  
"Byte prefix, NULL terminator"BZ
```

Escape Sequences for Four-Character Codes and String Chunks

The following escape sequences can be used in four-character codes and string chunks:

Escape Sequence	ASCII Value	Description
\n	10	Newline character
\t	9	Horizontal tab character
\b	8	Backspace character
\r	13	Carriage return character
\f	12	Form feed character
\\	92	Backslash
\'	39	Single quote
\"	34	Double quote
\ddd	Octal ddd	Arbitrary character

Extended Notation for Representing RIFF Form Definitions

To unambiguously define the structure of new RIFF forms, document the RIFF form using the basic notation along with the following extended notation:

Notation	Description
<name>	<p>A label that refers to some element of the file, where <name> is the name of the label. Examples follow:</p> <pre><NAME-ck> <GOBL-form> <bitmap-bits> <foo></pre> <p>Conventionally, a label that refers to a chunk is named <ckID-ck>, where 'ckID' is the chunk ID. Similarly, a label that refers to a RIFF form is named <formType-form>, where "formType" is the name of the form's type.</p>
<name> elements	<p>The actual data represented by <name> is defined as elements.</p> <p>This states that <name> is an abbreviation for elements, where elements is a sequence of other labels and literal data. An example follows:</p> <pre><GOBL-form> RIFF ('GOBL' <form-data>)</pre> <p>This example defines label <GOBL-form> as representing a RIFF form with chunk ID 'GOBL' and data equal to <form-data>, where <form-data> is a label that would be defined in another rule. Note that a label may represent any data, not just a RIFF chunk or form.</p>
<name:type>	<p>This is the same as <name>, but it also defines <name> to be equivalent to <type>. This notation obviates the following rule:</p> <pre><name> <type></pre>

This allows you to give a symbolic name to an element of a file format and to specify the element data type. An example follows:

```
<xyz-coordinate>    <x:INT> <y:INT> <z:INT>
```

This defines `<xyz-coordinate>` to consist of three parts concatenated together: `<x>`, `<y>`, and `<z>`. The definition also specifies that `<x>`, `<y>`, and `<z>` are integers. This notation is equivalent to the following:

```
<xyz-coordinate>    <x> <y> <z>
<x>    <INT>
<y>    <INT>
<z>    <INT>
```

[elements]

An optional sequence of labels and literal data. Surrounded by square brackets, it may be considered an element itself. An example follows:

```
<FOO-form>    RIFF('FOO' [ <header-ck> ] <data-ck> )
```

This example defines form "FOO" with an optional header chunk followed by a mandatory data chunk.

el1 | el2 | ... | elN

Exactly one of the listed elements must be present. An example follows:

```
<hdr-ck>    hdr( <hdr-x> | <hdr-y> | <hdr-z> )
```

This example defines the 'hdr' chunk's data as containing one of `<hdr-x>`, `<hdr-y>`, or `<hdr-z>`.

element...

One or more occurrences of **element** may be present. An ellipsis has this meaning only if it follows an element; in cases such as "el1|el2|...|elN," the ellipsis has its ordinary English meaning. If there is any possibility of confusion, an ellipsis should only be used to indicate one or more occurrences. An example follows:

```
<data-ck>    data( <count:INT> <item:INT> ... )
```

This example defines the data of the 'data' chunk to contain an integer `<count>`, followed by one or more occurrences of the integer `<item>`.

[element]...

Zero or more occurrences of **element** may be present. An example follows.

```
<data-ck>    data( <count:INT> [ <item:INT> ] ... )
```

This example defines the data of the 'data' chunk to contain an integer `<count>` followed by zero or more occurrences of an integer `<item>`.

{elements}

The group of elements within the braces should be considered a single element. An example follows:

```
<blorg>    <this> | { <that> | <other> } ...
```

This example defines `<blorg>` to be either `<this>` or one or more occurrences of `<that>` or `<other>`, intermixed in any way. Contrast this with the following example:

```
<blorg>    <this> | <that> | <other> ...
```

This example defines `<blorg>` to be either `<this>` or `<that>` or one or more occurrences of `<other>`.

struct { ...} name

A structure defined using C syntax. This can be used instead of a sequence of labels if a

C header (include) file is available that defines the structure. The label used to refer to the structure should be the same as the structure's typedef name. An example follows:

```
<3D_POINall struct {
    INT x;          /* X-coordinate */
    INT y;          /* Y-coordinate */
    INT z;          /* Z-coordinate */
} 3D_POINT
```

Because these types are more portable than C types such as int. The structure fields are assumed to be present in the file in the order given, with no padding or forced alignment.

Unless the RIFF chunk ID is 'RIFX', integer byte ordering is assumed to be in Intel format.

/* comment */

An explanatory comment to a rule. An example follows:

```
<weekend>    'Sat'|'Sun'    /* Four-character code */
                        /* for day */
```

A Sample RIFF Form Definition and RIFF Form

The following example defines <GOBL-form>, the hypothetical RIFF form of type 'GOBL'. To fully document a new RIFF form definition, a developer would also provide detailed descriptions of each file element, including the semantics of each chunk and sample files documented using the standard notation.

```
<GOBL-form >    RIFF ( 'GOBL'          /* RIFF form header */
                    [<org-ck>]          /* Origin chunk */
                    (default (0,0,0))  /*
                    <obj-list>)>        /* Series of graphical
                                        objects */

<org-ck>         org(    <origin:3D_POINT> )
                        /* Object-list origin */

<obj-list>       LIST(    'obj'    {    /* An object is a: */
                                <sqr-ck> |    /* square, */
                                <circ-ck> |    /* circle, */
                                <poly-ck> }.... ) /* or polygon */

<sqr-ck>         sqr(    <pt1:3D_POINT>    /* one vertex */
                        <pt2:3D_POINT>    /* another vertex */
                        <pt3:3D_POINT> )    /* a third vertex */

<circ-ck>        circ(    <center:3D_POINT>    /* Center of circle */
                        <circumPt:3D_POINT> )    /* Point on circumference */

<poly-ck>        poly( <pt:3D_POINT>... )    /* List of points in a polygon */

<3D_POINT>       struct          /* Defined in "gobl.h" */
{   INT x;          /* X-coordinate */
    INT y;          /* Y-coordinate */
    INT z;          /* Z-coordinate */
} 3D_POINT
```

Sample RIFF Form The following sample RIFF form adheres to the form definition for form type GOBL. The file contains three subchunks:

- An 'INFO' list
- An 'org' chunk
- An 'obj' chunk

The 'INFO' list and 'org' chunk each have two subchunks. The 'INFO' list is a registered global chunk that can be used within any RIFF file. The 'INFO' list is described in [INFO List Chunk](#).

Since the definition of the GOBL form does not refer to the INFO chunk, software that expects only 'org' and 'obj' chunks in a GOBL form

would ignore the unknown 'INFO' chunk.

```
RIFF( 'GOBL'
    LIST('INFO' /* INFO list containing file name and copyright */
        INAM("A House"Z)
        ICOP("(C) Copyright Encyclopedia International 1991"Z)
    )
    org(2, 0, 0) /* Origin of object list */
    LIST('obj' /* Object list containing two polygons */
        poly(0,0,0 2,0,0 2,2,0, 1,3,0, 0,2,0)
        poly(0,0,5 2,0,5 2,2,5, 1,3,5, 0,2,5)
    )
) /* End of form */
```

Storing Strings in RIFF Chunks

This section lists methods for storing text strings in RIFF chunks. While these guidelines may not make sense for all applications, you should follow these conventions if you must make an arbitrary decision regarding string storage.

NULL-Terminated String (ZSTR) Format

A NULL-terminated string (ZSTR) consists of a series of characters followed by a terminating NULL character. The ZSTR is better than a simple character sequence (STR) because many programs are easier to write if strings are NULL-terminated. ZSTR is preferred to a string with a size prefix (BSTR or WSTR) because the size of the string is already available as the **<ckSize>** value, minus one for the terminating NULL character.

String Table Format

In a string table, all strings used in a structure are stored at the end of the structure in packed format. The structure includes fields that specify the offsets from the beginning of the string table to the individual strings. An example follows:

```
typedef struct
{
    INT      iWidgetNumber; /* the widget number */
    USHORT   offszWidgetName; /* an offset to a string
                               in <rgchStrTab> */
    USHORT   offszWidgetDesc; /* an offset to a string
                               in <rgchStrTab> */
    INT      iQuantity; /* how many widgets */
    CHAR     rgchStrTab[1]; /* string table (allocate
                             as large as needed) */
} WIDGET;
```

If multiple chunks within the file need to reference variable-length strings, you can store the strings in a single chunk that acts as a string table. The chunks that refer to the strings contain offsets relative to the beginning of the data part of the string table chunk.

NULL-Terminated, Byte Size Prefix String (BZSTR) Series

In a BZSTR series, a series of strings is stored in packed format. Each string is a BZSTR, with a byte size prefix and a NULL terminator. This format retains the ease-of-use characteristics of the ZSTR while providing the string size, allowing the application to quickly skip

unnneeded strings.

Multiline String Format

When storing multiline strings, separate lines with a carriage return/line feed pair (ASCII 13/ASCII 10 pair). Although applications vary in their requirements for new line symbols (carriage return only, line feed only, or both), it is generally easier to strip out extra characters than to insert extra ones. Inserting characters might require reallocating memory blocks or pre-scanning the chunk before allocating memory for it.

Choosing a Storage Method

The following lists guidelines for deciding which storage method is appropriate for your application.

Usage	Recommended Format
Chunk data contains nothing except a string	ZSTR (NULL-terminated string) format.
Chunk data contains a number of fields, some of which are variable-length strings	String-table format
Multiple chunks within the file need to reference variable-length strings	String-table format
Chunk data stores a sequence of strings, some of which the application may want to skip	BZSTR (NULL-terminated string with byte size prefix) series
Chunk data contains multiline strings	A multiline string format

LIST Chunk

A LIST chunk contains a list, or ordered sequence, of subchunks. A LIST chunk is defined as follows:

```
LIST( <list-type> [<chunk>]... )
```

The **<list-type>** is a four-character code that identifies the contents of the list.

If an application recognizes the list type, it should know how to interpret the sequence of subchunks. However, since a LIST chunk may contain only subchunks (after the list type), an application that does not know about a specific list type can still walk through the sequence of subchunks.

Like chunk IDs, list types must be registered, and an all-lowercase list type has meaning relative to the form that contains it. See [Registering Multimedia Formats](#) for information on registering list types.

INFO List Chunk

The 'INFO' list is a registered global form type that can store information that helps identify the contents of the chunk. This information is useful but does not affect the way a program interprets the file; examples are copyright information and comments. An 'INFO' list is a 'LIST' chunk with list type 'INFO'. The following shows a sample 'INFO' list chunk:

```
LIST('INFO'      INAM("Two Trees"Z)
      ICMT("A picture for the opening screen"Z) )
```

An 'INFO' list should contain only the following chunks. New chunks may be defined, but an application should ignore any chunk it doesn't understand. The chunks listed below may only appear in an 'INFO' list. Each chunk contains a ZSTR, or null-terminated text string.

Chunk ID	Description
IARL	<i>Archival Location.</i> Indicates where the subject of the file is archived.
IART	<i>Artist.</i> Lists the artist of the original subject of the file. For example, "Michaelangelo."
ICMS	<i>Commissioned.</i> Lists the name of the person or organization that commissioned the subject of the file. For example, "Pope Julian II."
ICMT	<i>Comments.</i> Provides general comments about the file or the subject of the file. If the comment is several sentences long, end each sentence with a period. Do <i>not</i> include newline characters.
ICOP	<i>Copyright.</i> Records the copyright information for the file. For example, "Copyright Encyclopedia International 1991." If there are multiple copyrights, separate them by a semicolon followed by a space.
ICRD	<i>Creation date.</i> Specifies the date the subject of the file was created. List dates in year-month-day format, padding one-digit months and days with a zero on the left. For example, "1553-05-03" for May 3, 1553.
ICRP	<i>Cropped.</i> Describes whether an image has been cropped and, if so, how it was cropped. For example, "lower right corner."
IDIM	<i>Dimensions.</i> Specifies the size of the original subject of the file. For example, "8.5 in h, 11 in w."
IDPI	<i>Dots Per Inch.</i> Stores dots per inch setting of the digitizer used to produce the file, such as "300."
IENG	<i>Engineer.</i> Stores the name of the engineer who worked on the file. If there are multiple engineers, separate the names by a semicolon and a blank. For example, "Smith, John; Adams, Joe."
IGNR	<i>Genre.</i> Describes the original work, such as, "landscape," "portrait," "still life," etc.
IKEY	<i>Keywords.</i> Provides a list of keywords that refer to the file or subject of the file. Separate multiple keywords with a semicolon and a blank. For example, "Seattle; aerial view; scenery."
ILGT	<i>Lightness.</i> Describes the changes in lightness settings on the digitizer required to produce the file. Note that the format of this information depends on hardware used.
IMED	<i>Medium.</i> Describes the original subject of the file, such as, "computer image," "drawing," "lithograph," and so forth.
INAM	<i>Name.</i> Stores the title of the subject of the file, such as, "Seattle From Above."
IPLT	<i>Palette Setting.</i> Specifies the number of colors requested when digitizing an image, such as "256."
IPRD	<i>Product.</i> Specifies the name of the title the file was originally intended for, such as "Encyclopedia of Pacific Northwest Geography."
ISBJ	<i>Subject.</i> Describes the contents of the file, such as "Aerial view of Seattle."

ISFT	<i>Software.</i> Identifies the name of the software package used to create the file, such as "Microsoft WaveEdit."
ISHP	<i>Sharpness.</i> Identifies the changes in sharpness for the digitizer required to produce the file (the format depends on the hardware used).
ISRC	<i>Source.</i> Identifies the name of the person or organization who supplied the original subject of the file. For example, "Trey Research."
ISRF	<i>Source Form.</i> Identifies the original form of the material that was digitized, such as "slide," "paper," "map," and so forth. This is not necessarily the same as IMED.
ITCH	<i>Technician.</i> Identifies the technician who digitized the subject file. For example, "Smith, John."

CSET (Character Set) Chunk

To define character-set and language information for a RIFF file, use the CSET chunk. The CSET chunk defines the code page and country, language, and dialect codes for the file. These values can be overridden for specific file elements; see [Usage Codes for Extra Header and Extra Entry Fields](#) for information on specifying character set information in a compound file.

The CSET chunk is defined as follows:

```
<CSET chunk>    CSET( <usCodePage:USHORT>
                  <usCountryCode:USHORT>
                  <usLanguageCode:USHORT>
                  <usDialect:USHORT> )
```

The fields are as follows:

Field	Description
<i>usCodePage</i>	Specifies the code page used for file elements. If the CSET chunk is not present, or if this field has value zero, assume standard ISO 8859/1 code page (identical to code page 1004 without code points defined in hex columns 0, 1, 8, and 9).
<i>usCountryCode</i>	Specifies the country code used for file elements. See Country Codes , for a list of currently defined country codes. If the CSET chunk is not present, or if this field has value zero, assume USA (country code 001).
<i>usLanguage, usDialect</i>	Specify the language and dialect used for file elements. See Language and Dialect Codes for a list of language and dialect codes. If the CSET chunk is not present, or if these fields have value zero, assume US English (language code 9, dialect code 1).

Country Codes

Use one of the following country codes in the *usCountryCode* field:

Country Code	Country
000	None (ignore this field)
001	USA

002	Canada
003	Latin America
030	Greece
031	Netherlands
032	Belgium
033	France
034	Spain
039	Italy
041	Switzerland
043	Austria
044	United Kingdom
045	Denmark
046	Sweden
047	Norway
049	West Germany
052	Mexico
055	Brazil
061	Australia
064	New Zealand
081	Japan
082	Korea
086	People's Republic of China
088	Taiwan
090	Turkey
351	Portugal
352	Luxembourg
354	Iceland
358	Finland

Language and Dialect Codes

Specify one of the following pairs of language-code and dialect-code values in the *usLanguage* and *usDialect* fields:

Language Code	Dialect Code	Language
0	0	None (ignore these fields)
1	1	Arabic
2	1	Bulgarian

3	1	Catalan
4	1	Traditional Chinese
4	2	Simplified Chinese
5	1	Czech
6	1	Danish
7	1	German
7	2	Swiss German
8	1	Greek
9	1	US English
9	2	UK English
10	1	Spanish
10	2	Spanish Mexican
11	1	Finnish
12	1	French
12	2	Belgian French
12	3	Canadian French
12	4	Swiss French
13	1	Hebrew
14	1	Hungarian
15	1	Icelandic
16	1	Italian
16	2	Swiss Italian
17	1	Japanese
18	1	Korean
19	1	Dutch
19	2	Belgian Dutch
20	1	Norwegian - Bokmal
20	2	Norwegian - Nynorsk
21	1	Polish
22	1	Brazilian Portuguese
22	2	Portuguese
23	1	Rhaeto-Romanic
24	1	Romanian
25	1	Russian
26	1	Serbo-Croatian (Latin)
26	2	Serbo-Croatian (Cyrillic)
27	1	Slovak
28	1	Albanian

29	1	Swedish
30	1	Thai
31	1	Turkish
32	1	Urdu
33	1	Bahasa

JUNK (Filler) Chunk

A JUNK chunk represents padding, filler or outdated information. It contains no relevant data; it is a space filler of arbitrary size. The JUNK chunk is defined as follows:

```
<JUNK chunk>    JUNK( <filler> )
```

where **<filler>** contains random data.

Compound File Structure

The compound file structure is a RIFF-based structure upon which multimedia file formats can be defined. The compound file structure is a parameterized structure that provides for the following:

- Storage of multimedia data elements
- Direct access to multimedia data elements (as opposed to sequential searching)

The goals of the compound file structure are to maximize flexibility and extensibility while minimizing implementation costs. Using the compound file structure, developers of multimedia data formats can define both simple and complex file formats.

The structure is flexible enough to be used for many purposes, but it can be simplified for use with simple file formats. Designers of new multimedia file formats can restrict the use of standard header fields, requiring some and removing others.

For example, a developer might define a compound file format that stores a series of bit maps in a single file, thus reducing compact disc seek times. Another developer might define a compound file format that contains a special type of audio resource, using the compound file header information to identify the attributes of the audio data stored within.

Structural Overview

Files based upon the compound file structure contain the following two RIFF chunks at their topmost level:

- Compound File Table of Contents (CTOC) chunk
- Compound File Element Group (CGRP) chunk

The CTOC chunk indexes the CGRP chunk, which contains the actual multimedia data elements. Defined using the standard chunk notation, a compound file is represented as follows:

```
<compound file>    RIFF('type' <CTOC> <CGRP>)
```

where 'type' is a FOURCC indicating the file type.

This section describes the CTOC and CGRP chunks in detail.

Compound File Table of Contents (CTOC) Chunk

The CTOC chunk functions mainly as an index, allowing direct access to elements within a compound file. The CTOC chunk also contains information about the attributes of the entire file and of each media element within the file.

To provide the maximum flexibility for defining compound file formats, the CTOC chunk can be customized at several levels. The CTOC chunk contains fields whose length and usage is defined by other CTOC fields. This parameterization adds complexity, but it provides flexibility to file format designers and allows applications to correctly read data without necessarily knowing the specific file format definition.

Structural Overview

The CTOC chunk defines the contents of the CGRP chunk. The CTOC chunk has the following components:

- Header information defining the size of the CTOC chunk, the number of entries in the CGRP chunk, the size of the CGRP chunk, and general information about the entire header file
- A parameter table definition defining the size and contents of the header parameter table and CTOC table entries
- A header parameter table defining attributes that apply to the entire compound file.
- CTOC table entries defining the location, size, name, and attributes of the compound file elements contained in the CGRP chunk.

These components appear sequentially in the CTOC chunk. The individual fields in the CTOC chunk can be found by looking under [MMCFINFO](#) or [MMCTOCENTRY](#) respectively.

Following are lists of each area of fields.

Header Information

The header information section defines general information about the CTOC header and about the entire compound file. It contains the following fields:

- *ulHeaderSize*
 - *ulEntriesTotal*
 - *ulEntriesDeleted*
 - *ulEntriesUnused*
 - *ulBytesTotal*
 - *ulBytesDeleted*
 - *ulHeaderFlags*
-

Parameter Table Definition

The parameter table definition defines the size and contents of the header parameter table and CTOC table. It contains the following fields:

- *usEntrySize*
- *usNameSize*
- *usExHdrFields*
- *usExEntFields*

- *aulExHdrFldUsage*
- *aulExEntFldUsage*

Valid usage codes for each field in this array are listed in [Usage Codes for Extra Header and Extra Entry Fields](#).

Header Parameter Table

The header parameter table is an optional component generally used to define attributes of the entire compound file.

- *aulExHdrField*

CTOC Table Entries

The CTOC table entries define the location, size, name, and other information about the individual compound file elements contained in the CGRP chunk. The number of CTOC table entries is determined by the *ulEntriesTotal* field in the header information of the CTOC chunk.

Each CTOC table entry is a structure containing the following fields:

- *ulOffset*
- *ulSize*
- *ulMedType*
- *ulMedUsage*
- *ulCompressTech*
- *ulUncompressBytes*
- *aulExEntField*
- *pszElementName*

Usage Codes for Extra Header and Extra Entry Fields

The following are valid usage codes for elements in the *aulExHdrFldUsage* and *aulExEntFldUsage* arrays, both of which are fields of the CTOC header. These arrays define the meaning of data stored in the *aulExHdrField* and *aulExEntField* "extra fields." All usage codes apply to both header fields and entry fields, unless explicitly stated otherwise.

Values marked in the extra header field arrays generally apply to all elements in the CFRG chunk, while values marked in the extra entry field arrays generally apply only to the element referenced by the corresponding CTOC table entry.

CTOC_EFU_UNUSED (0x00)

The field is unused. This usage code may be used to logically delete a header field.

CTOC_EFU_LASTMODTIME (0x01)

When used to describe an extra header field, the field contains the time that any portion of the CTOC or CGRP was last modified.

When used to describe an extra entry field, the field contains the time that the corresponding CTOC table entry, or the compound file element it refers to, was last modified.

The field is interpreted as a ULONG containing the number of seconds that have elapsed since 00:00:00 Greenwich Mean Time (GMT), January 1, 1970.

CTOC_EFU_CODEPAGE

The field contains the code page and country code for the *achName* field. These values override any values specified in a CSET chunk.

When used to describe an extra header field, the field contains code-page and country-code information for all CTOC table entries. When used to describe an extra entry field, the field contains information for that specific CTOC table entry.

The low-order word of the field contains one of the following code page values:

Zero

Use standard ISO 8859/1 code page. This is identical to code page 1004 without code points defined in hex columns 0, 1, 8, and 9.

CTOC_CHARSET_CODEPAGE (0x0000 *nnnn*)

Use code page 0x*nnnn*, where 0x*nnnn* is the 16-bit code page number. For example, 0x00000352 for OS/2 code page 850, or 0x000004E4 for Windows 3.1 code page 1252.

The high-order word contains one of the following country codes:

Zero

Ignore this field.

Country code

See [Country Codes](#) for a list of currently defined country codes.

CTOC_EFU_LANGUAGE

The field contains language and dialect information for the *achName* field. These values override any values specified in a CSET chunk.

When used to describe an extra header field, the field contains language information for all CTOC table entries. When used to describe an extra entry field, the field contains information for that specific CTOC table entry.

The low-order word of the field contains one of the following language codes:

Zero

Ignore this field.

Language code

See [Language and Dialect Codes](#) for a list of currently defined language codes.

The high-order word of the field contains one of the following dialect codes:

Zero

Ignore this field.

Dialect code

Select [Language and Dialect Codes](#) for a list of currently defined dialect codes.

CTOC_EFU_COMPRESSPARAM0 (0x05) through CTOC_EFU_COMPRESSPARAM9 (0x14)

Specifies a compression parameter. See [Compression of Compound File Elements](#).

Compression of Compound File Elements

Compound file elements can be compressed. The *uiCompressTech* field of a CTOC table entry contains a FOURCC compression technique identifier for the corresponding compound file element. If the field is zero, the compound file element is not compressed.

The definition of a specific compression technique may specify that either the entire compound file element is compressed, or that some specific subset, for example one or more RIFF chunks, is compressed.

The *uiUncompressSize* field contains the number of bytes that the compound file element will occupy in memory after decompression. If the compound file element is not compressed, this field contains the same value as the *uiSize* field, which identifies the file size of the compound file element.

Compression techniques may demand extra header fields or extra entry fields for decompression parameters. Compression technique

identifiers, and any new entry fields corresponding to decompression technique parameters, must be unique. See [Registering Multimedia Formats](#) for registration information.

Compound File Element Group (CGRP) Chunk

The actual elements of data referenced by the CTOC chunk are stored in a compound file Element Group (CGRP) chunk. The CGRP chunk contains all the compound file elements, concatenated together into one contiguous block of data. Some of the elements in the CGRP chunk might be unused, if the element was marked for deletion or was altered and stored elsewhere within the CGRP chunk.

Elements within the CGRP chunk are of arbitrary size and can appear in a specific or arbitrary order, depending upon the file format definition. Each element is identified by a corresponding CTOC table entry.

Using the standard RIFF notation, the CGRP chunk is defined as follows:

```
<CGRP-chunk>    CGRP([<compound file element>]...)
```

Placement of the CTOC and CGRP Chunks

The specific file format definition can specify which of the two chunks appear first in the data file. Generally, the CTOC chunk is placed at the front of the file to reduce the seek and read times required to access it. During authoring time, an application might place the CTOC chunk at the end of the file, so it can be expanded as elements are added to the CGRP chunk.

Multimedia File Formats

This appendix describes the following multimedia file formats:

- Audio/Video Interleaved (AVI)
- Bundle (BND)
- Device-independent bitmap (DIB)
- RIFF DIB (RDIB)
- Musical Instrument Digital Interface (MIDI)
- RIFF MIDI (RMID)
- Palette (PAL)
- Rich Text Format (RTF)
- Waveform audio (WAVE)

Most of these file formats are based on the resource interchange file format (RIFF), described in the *OS/2 Multimedia Application Programming Guide*.

Audio/Video Interleaved (AVI) Format

The Microsoft Audio/Video Interleaved (AVI) file format is a RIFF file specification used with applications that capture, edit, and playback audio/video sequences. In general, AVI files can contain multiple streams of different types of data. Most AVI sequences will use a single audio and a single video stream. A simple variation for an AVI sequence uses a single video stream and does not contain an audio stream.

This section describes the types of AVI files supported by OS/2 multimedia. Refer to the Microsoft documentation for a complete description of the AVI file format.

AVI RIFF Form

AVI files use the AVI RIFF form. The AVI RIFF form is identified by the four-character code 'AVI' RIFF form type. All AVI files include two mandatory LIST chunks. These chunks define the format of the streams and stream data. AVI files generally include an index chunk. This optional chunk specifies the location of data chunks within the file. An AVI file with these components has the following form:

```
RIFF ('AVI'
  LIST ('hdrl'
    .
    .
    .
  )
  LIST ('movi'
    .
    .
    .
  )
  [ 'idx1' <AVI Index> ]
)
```

The LIST chunks and the index chunk are subchunks of the RIFF 'AVI' chunk. The 'AVI' chunk identifies the file as an AVI RIFF file. The LIST 'hdrl' chunk defines the format of the data and is the first required list chunk. The LIST 'movi' chunk contains the data for the AVI sequence and is the second required list chunk. The 'idx1' chunk is the optional index chunk. AVI files must keep these three components in the proper sequence.

The LIST 'hdrl' and LIST 'movi' chunks use subchunks for their data. The following example shows the AVI RIFF form expanded with the chunks needed to complete the LIST 'hdrl' and LIST 'movi' chunks:

```
RIFF ('AVI'
  LIST ('hdrl'
    'avih'(<Main AVI Header>)
    LIST ('strl'
      'strl'(<Stream header>)
      'strf'(<Stream format>)
      'strd'(additional header data)
      .
      .
      .
    )
    .
    .
    .
  )
  LIST ('movi'
    {SubChunk | LIST ('rec'
      SubChunk1
      SubChunk2
      .
      .
      .
    )
    .
    .
    .
  )
  [ 'idx1' <AVIIndex> ]
)
```

The following sections describe the chunks contained in the LIST 'hdrl' and LIST 'movi' chunks as well as the 'idx1' chunk.

Data Structures for AVI Files

Data structures used in the RIFF chunks are defined in the AVIFMT.H header file. The reference section which follows describes the data structures that are used for the main AVI header, stream headers and AVI index chunks.

The Main AVI Header LIST

The file begins with the main header. In the AVI file, this header is identified with an 'avih' four-character code. The header contains general information about the file, such as the number of streams within the file and the width and height of the AVI sequence. The main header has the following data structure defined for it:

```
typedef struct {
    ULONG ulMicroSecPerFrame;
    ULONG ulMaxBytesPerSec;
    ULONG ulReserved1;
    ULONG ulFlags;
    ULONG ulTotalFrames;
    ULONG ulInitialFrames;
    ULONG ulStreams;
    ULONG ulSuggestedBufferSize;
    ULONG ulWidth;
    ULONG ulHeight;
    ULONG ulReserved[4];
} MainAVIHeader;
```

The *ulMicroSecPerFrame* field specifies the period between video frames. This value indicates the overall timing for the file.

The *ulMaxBytesPerSec* field specifies the approximate maximum data rate of the file. This value indicates the number of bytes per second the system must handle to present an AVI sequence as specified by the other parameters contained in the main header and stream header chunks.

The *ulFlags* field contains any flags for the file. The AVIF_HASINDEX flag applies to files with an index chunk. The AVI_HASINDEX flag indicates an index is present. The AVIF_ISINTERLEAVED flag indicates the AVI file has been interleaved. The system can stream interleaved data more efficiently than non-interleaved data.

The *ulTotalFrames* field of the main header specifies the total number of frames of data in file.

The *ulInitialFrames* field is used for some interleaved files. If you are creating interleaved files with audio skewing, specify the number of audio frames in the file prior to the initial video frame of the AVI sequence in this field.

The *ulStreams* field specifies the number of streams in the file. For example, a file with audio and video has two (2) streams.

The *ulSuggestedBufferSize* field specifies the suggested buffer size for reading the file. Generally, this size should be large enough to contain the largest chunk in the file. For an interleaved file, the buffer size should be large enough to read an entire record and not just a chunk.

The *ulWidth* and *ulHeight* fields specify the width and height of the AVI file in pixels.

The Stream Header 'strl' Chunks

The main header is followed by one or more 'strl' chunks. A 'strl' chunk is required for each data stream. These chunks contain information about the streams in the file. Each 'strl' chunk must contain a stream header and stream format chunk. Stream header chunks are identified by the 'strh' four-character code and stream format chunks are identified by the 'strf' four-character code. In addition to the stream header and stream format chunks, the 'strl' chunk might also contain a stream data chunk. Stream data chunks are identified with the four-character code 'strd'.

The stream header has the following data structure defined for it:

```
typedef struct {
    FOURCC fccType;
    FOURCC fccHandler;
    ULONG ulFlags;
    ULONG ulReserved1;
```

```

    ULONG    ulInitialFrames;
    ULONG    ulScale;
    ULONG    ulRate;
    ULONG    ulStart;
    ULONG    ulLength;
    ULONG    ulSuggestedBufferSize;
    ULONG    ulQuality;
    ULONG    ulSampleSize;
    ULONG    ulReserved[2];
} AVIStreamHeader;

```

The stream header specifies the type of data the stream contains, such as audio or video, by means of a four-character code. The *fccType* field is set to 'vids' if the stream it specifies contains video data. It is set to 'auds' if it contains audio data.

The *fccHandler* field contains a four-character code describing the installable decompressor used with the data.

The *ulFlags* field contains any flags for the data stream. The AVISF_DISABLED flag indicates that the stream data should be rendered only when explicitly enabled by the user.

The *ulInitialFrames* is used for interleaved files. If you are creating interleaved files with audio skewing, specify the number of audio frames in the file prior to the initial video frame of the AVI sequence in this field.

The remaining fields describe the playback characteristics of the stream. These factors include the playback rate (*ulScale* and *ulRate*), the starting time of the sequence (*ulStart*), the length of the sequence (*ulLength*), the size of the playback buffer (*ulSuggestedBuffer*), an indicator of the data quality (*ulQuality*), and sample size (*ulSampleSize*). See the reference section for more information on these fields.

Some of the fields in the stream header structure are also present in the main header structure. The data in the main header structure applies to the whole file while the data in the stream header structure applies only to a stream.

A stream format 'strf' chunk must follow a stream header 'strh' chunk. The stream format chunk describes the format of the data in the stream. For video streams, the information in this chunk is a Windows BITMAPINFO structure (including palette information if appropriate). For audio streams, the information in this chunk is a Windows WAVEFORMATX or PCMWAVEFORMAT structure.

The 'strl' chunk might also contain a stream data 'strd' chunk. If used, this chunk follows the stream format chunk. The format and content of this chunk is defined by installable decompressors. Typically, decompressors use this information for configuration. Applications that read and write RIFF files do not need to decode this information. They transfer this data to and from a decompressor as a memory block.

An AVI player associates the stream headers in the LIST 'hdrl' chunk with the stream data in the LIST 'movi' chunk by using the order of the 'strl' chunks. The first 'strl' chunk applies to stream 0, the second applies to stream 1, and so forth. For example, if the first 'strl' chunk describes the video data, the video data is contained in stream 0. Similarly, if the second 'strl' chunk describes audio data, then the audio data is contained in stream 1.

The LIST 'movi' Chunk

Following the header information is a LIST 'movi' chunk that contains chunks of the actual data in the streams; that is, the pictures and sounds themselves. The data chunks are grouped into 'rec' chunks.

Like any RIFF chunk, the data chunks contain a four-character code to identify the chunk type. The four-character code that identifies each chunk consists of the stream number and a two-character code that defines the type of information encapsulated in the chunk. For example, a waveform chunk is identified by a of 'wb' two-character code. If a waveform chunk corresponded to the second LIST 'hdrl' stream description, it would have a '01wb' four-character code.

Since all the format information is in the header, the audio data contained in these data chunks does not contain any information about its format. An audio data chunk has the following format (the ## in the format represents the stream identifier):

```

WAVE Bytes    '##wb'
      BYTE    abBytes[];

```

Video data can be compressed or uncompressed DIBs. An uncompressed DIB has BI_RGB specified for the *biCompression* field in its associated BITMAPINFO structure. A compressed DIB has a value other than BI_RGB specified in the *biCompression* field.

A data chunk for an uncompressed DIB contains RGB video data. These chunks are identified with a two-character code of "db" (db is an abbreviation for DIB bits). Data chunks for a compressed DIB are identified with a two-character code of 'dc' (dc is an abbreviation for DIB compressed). Neither data chunk will contain any header information about the DIBs. The data chunk for an uncompressed DIB has the following form:

```
DIB Bits    '##db'
  BYTE      abBits[ ];
```

The data chunk for a compressed DIB has the following form:

```
Compressed DIB  '##dc'
  BYTE          abBits[ ];
```

The 'idx1' Chunk

AVI files can have an index chunk after the LIST 'movi' chunk. The index chunk essentially contains a list of the data chunks and their location in the file. This provides efficient random access to the data within the file, because an application can locate a particular sound sequence or video image in a AVI file without having to scan it.

Index chunks use the 'idx1' four-character code. The following data structure is defined for index entries:

```
typedef struct {
    ULONG    ckid;
    ULONG    ulFlags;
    ULONG    ulChunkOffset;
    ULONG    ulChunkLength;
} AVIINDEXENTRY;
```

The *ckid*, *ulFlags*, *ulChunkOffset*, and *ulChunkLength* entries are repeated in the AVI file for each data chunk indexed. The index will have entries for each 'rec' chunk. The 'rec' entries should have the AVIIF_LIST flag set and the list type in the *ckid* field.

The *ckid* field identifies the data chunk. This field uses four-character codes for the identifying chunk.

The *ulFlags* field specifies any flags for the data. The AVIIF_KEYFRAME flag indicates key frames in the video sequence. Key frames do not need previous video information to be decompressed. The AVIIF_NOTIME flag indicates a chunk does not affect the timing of a video stream. The AVIIF_LIST flag indicates the current chunk is a LIST chunk. Use the *ckid* field to identify the type of LIST chunk.

The *ulChunkOffset* and *ulChunkLength* fields specify the position of the chunk and the length of the chunk. The *ulChunkOffset* field specifies the position of the chunk in the file relative to the 'movi' list. The *ulChunkLength* field specifies the length of the chunk excluding the eight bytes for the RIFF header.

If you include an index in the RIFF file, set the AVIF_HASINDEX in the *ulFlags* field of the AVI header. (This header is identified by 'avih' chunk ID.) This flag indicates that the file has an index.

Other Data Chunks

If you need to align data in you AVI file you can add a 'JUNK' chunk. (This chunk is a standard RIFF type.) Applications reading these chunks will ignore their contents. Files played from CD-ROM can use these chunks to align data so they can be read more efficiently. You might want to use this chunk to align your data for the 2 kilobyte CD-ROM boundaries. The 'JUNK' chunk has the following form:

```
AVI Padding    'JUNK'
  Byte          data[ ]
```

As with any other RIFF files, all applications that read AVI files should ignore the non-AVI chunks that it does not recognize. Applications that read and write AVI files should preserve the non-AVI chunk when they save files they have loaded.

Interleaved Files

All AVI files produced by OS/2 multimedia are interleaved. The audio stream is divided into single frame pieces. The video and audio data for each frame are grouped into 'rec' chunks.

Platforms, other than OS/2 multimedia, have additional requirements to playback from CD-ROM devices. When OS/2 multimedia creates a file with the moderate frame sizes and frame rates which these other platforms support, it performs two additional steps to accommodate the needs of these platforms:

1. Audio skewing
- The audio data is skewed ahead of the video data by approximately 0.75 seconds. For example, a 15 frame per second movie would have 12 audio frames skewed ahead of the first video frame. The *ulInitialFrames* fields in the main and video stream headers are set to the number of skewed frames.
2. Padding
- The 'rec' chunks are padded so that their size is a multiple of 2 kilobytes and so that the beginning of the actual data in the LIST chunk lies on a 2 kilobyte boundary.

OS/2 multimedia does not require either padding or skewing to playback from CD-ROM. However, to maintain compatibility with platforms, skewing and padding are performed on files with moderate data rates. Movie files with data rates less than or equal to the nominal data rate of an uncompressed 15 frame per second movie with a frame size of 160 by 120 pels are skewed and padded. This allows other platforms to playback these files.

However, OS/2 multimedia supports movies with nominal frame sizes of 320 by 240 pels at 15 frames per second. CD-ROM data rates do not permit the wasted bandwidth required by padding at this large frame rate. Files with these large data rate requirements are not padded.

AVI RIFF File Reference

This section lists data structures used to support AVI RIFF files. These structures are defined in AVIFMT.H. The data structures are presented in alphabetical order. The structure definition is given, followed by a description of each field.

AVIINDEXENTRY

The AVI file index consists of an array of **AVIINDEXENTRY** structures contained within an 'idx1' chunk at the end of an AVI file. This chunk follows the main LIST 'movi' chunk which contains the actual data.

```
typedef struct {
    ULONG      ckid;
    ULONG      ulFlags;
    ULONG      ulChunkOffset;
    ULONG      ulChunkLength;
} AVIINDEXENTRY;
```

The **AVIINDEXENTRY** structure has the following fields:

Field	Description
<i>ckid</i>	Specifies a four-character code corresponding to the chunk ID of a data chunk in the file.
<i>ulFlags</i>	Specifies any applicable flags. The flags in the low-order word are reserved for AVI, while those in the high-order word can be used for stream- and compressor/decompressor-specific information. The following values are currently defined: AVIIF_LIST AVIIF_KEYFRAME
	Indicates the specified chunk is a 'LIST' chunk, and the <i>ckid</i> field contains the list type of the chunk. Indicates this chunk is a key frame. Key frames do not require additional preceding chunks to be properly

	decoded.
AVIIF_FIRSTPART	
	Indicates this chunk needs the frames following it to be used; it cannot stand alone.
AVIIF_LASTPART	
	Indicates this chunk needs the frames preceding it to be used; it cannot stand alone.
AVIIF_NOTIME	
	Indicates this chunk should have no effect on timing or calculating time values based on the number of chunks. For example, palette change chunks in a video stream should have this flag set, so that they are not counted as taking up a frame's worth of time.
<i>ulChunkOffset</i>	Specifies the position in the file of the specified chunk. The position value includes the eight byte RIFF header.
<i>ulChunkLength</i>	Specifies the length of the specified chunk. The length value does not include the eight byte RIFF header.

AVIStreamHeader

The **AVIStreamHeader** structure contains header information for a single stream of a file. It is contained within an 'strh' chunk within a LIST 'strl' chunk that is itself contained within the LIST 'hdrl' chunk at the beginning of an AVI RIFF file.

```
typedef struct {
    FOURCC    fccType;
    FOURCC    fccHandler;
    ULONG     ulFlags;
    ULONG     ulReserved1;
    ULONG     ulInitialFrames;
    ULONG     ulScale;
    ULONG     ulRate;
    ULONG     ulStart;
    ULONG     ulLength;
    ULONG     ulSuggestedBufferSize;
    ULONG     ulQuality;
    ULONG     ulSampleSize;
    ULONG     Reserved[2];
} AVIStreamHeader;
```

The **AVIStreamHeader** structure has the following fields:

Field	Description				
<i>fccType</i>	<p>Contains a four-character code which specifies the type of data contained in the stream. The following values are currently defined for AVI data:</p> <table> <tr> <td>'vids'</td><td>Indicates the stream contains video data. The stream format chunk contains a BITMAPINFO structure which can include palette information.</td></tr> <tr> <td>'auds'</td><td>Indicates the stream contains audio data. The stream format chunk contains a WAVEFORMAT or PCMWAVEFORMAT structure.</td></tr> </table> <p>Other four-character codes can identify non-AVI data.</p>	'vids'	Indicates the stream contains video data. The stream format chunk contains a BITMAPINFO structure which can include palette information.	'auds'	Indicates the stream contains audio data. The stream format chunk contains a WAVEFORMAT or PCMWAVEFORMAT structure.
'vids'	Indicates the stream contains video data. The stream format chunk contains a BITMAPINFO structure which can include palette information.				
'auds'	Indicates the stream contains audio data. The stream format chunk contains a WAVEFORMAT or PCMWAVEFORMAT structure.				
<i>fccHandler</i>	Optionally, contains a four-character code that identifies a specific data handler. The data handler is the preferred handler for the stream.				

ulFlags

Specifies any applicable flags. The bits in the high-order word of these flags are specific to the type of data contained in the stream. The following flags are currently defined:

AVISF_DISABLED

Indicates this stream should not be enabled by default.

AVISF_VIDEO_PALCHANGES

Indicates this video stream contains palette changes. This flag warns the playback software that it will need to animate the palette.

ulReserved1

Reserved. (Should be set to 0.)

ulInitialFrames

Specifies how far audio data is skewed ahead of the video frames in interleaved files. Typically, this is about 0.75 seconds.

ulScale

This field is used together with *ulRate* to specify the time scale that this stream will use.

Dividing *ulRate* by *ulScale* gives the number of samples per second.

For video streams, this rate should be the frame rate.

For audio streams, this rate should correspond to the time needed for *nBlockAlign* bytes of audio, which for PCM audio simply reduces to the sample rate.

ulRate

See *ulScale*.

ulStart

This field is currently reserved and should be set to zero.

ulLength

Specifies the length of this stream. The units are defined by the *ulRate* and *ulScale* fields of the stream's header.

ulSuggestedBufferSize

Suggests how large a buffer should be used to read this stream. Typically, this contains a value corresponding to the largest chunk presented in the stream. Using the correct buffer size makes playback more efficient. Use zero if you do not know the correct buffer size.

ulQuality

Specifies an indicator of the quality of the data in the stream. Quality is represented as a number between 0 and 10000. For compressed data, this typically represent the value of the quality parameter passed to the compression software. If set to -1, drivers use the default quality value.

ulSampleSize

Specifies the size of a single sample of data. This is set to zero if the samples can vary in size. If this number is non-zero, then multiple samples of data can be grouped into a single chunk within the file. If it is zero, each sample of data (such as a video frame) must be in a separate chunk.

For video streams, this number is typically zero, although it can be non-zero if all video frames are the same size.

For audio streams, this number should be the same as the *nBlockAlign* field of the **WAVEFORMAT** structure describing the audio.

MainAVIHeader

The **MainAVIHeader** structure contains global information for the entire AVI file. It is contained within an 'avih' chunk within the LIST 'hdr1' chunk at the beginning of an AVI RIFF file.

```
typedef struct {
    ULONG    ulMicroSecPerFrame;
    ULONG    ulMaxBytesPerSec;
    ULONG    ulReserved1;
```



```

    ULONG    ulFlags;
    ULONG    ulTotalFrames;
    ULONG    ulInitialFrames;
    ULONG    ulStreams;
    ULONG    ulSuggestedBufferSize;
    ULONG    ulWidth;
    ULONG    ulHeight;
    ULONG    ulReserved {4}
} MainAVIHeader;

```

The **MainAVIHeader** structure has the following fields:

Field	Description
<i>ulMicroSecPerFrame</i>	Specifies the number of microseconds between frames.
<i>ulMaxBytesPerSec</i>	Specifies the approximate maximum data rate of file.
<i>ulReserved1</i>	Reserved. (This field should be set to 0.)
<i>ulFlags</i>	Specifies any applicable flags. The following flags are defined: <div> <div>AVIF_HASINDEX</div> <div>Indicates the AVI file has an 'idx1' chunk containing an index at the end of the file. For good performance, all AVI files should contain an index.</div> </div> <div> <div>AVIF_ISINTERLEAVED</div> <div>Indicates the AVI file is interleaved.</div> </div>
<i>ulTotalFrames</i>	Specifies the number of frames of data in file.
<i>ulInitialFrames</i>	Specifies the initial frame for interleaved files. Non-interleaved files should specify zero.
<i>ulStreams</i>	Specifies the number of streams in the file. For example, a file with audio and video has two (2) streams.
<i>ulSuggestedBufferSize</i>	Specifies the suggested buffer size for reading the file. Generally, this size should be large enough to contain the largest chunk in the file. If set to zero, or if it is too small, the playback software will have to reallocate memory during playback which will reduce performance. <div> <div>For an interleaved file, this buffer size should be large enough to read an entire record and not just a chunk.</div> </div>
<i>ulWidth</i>	Specifies the width of the AVI file in pixels.
<i>ulHeight</i>	Specifies the height of the AVI file in pixels.
<i>ulReserved</i>	Reserved. These four double words should be set to zero.

Bundle File Format

The bundle (BND) format contains a series of RIFF chunks or other multimedia files. The BND file is defined as follows:

```
<BND-file>    RIFF('BND' <CTOC-chunk> <CGRP-chunk> )
```

The **<CTOC-chunk>** and **<CGRP-chunk>** formats are defined in the *OS/2 Multimedia Application Programming Guide*.

Each compound file element must be capable of standing alone as an independent file. An element cannot be a random chunk (except the RIFF chunk, indicating a RIFF file) or random binary data (unless the binary data is to be treated as a file).

Device-Independent Bitmap File Format

The device-independent bitmap (DIB) format represents bitmap images in a device-independent manner. Bitmaps can be represented at 1, 4, and 8 bits per pixel, with a palette containing colors represented in 24 bits. Bitmaps also can be represented at 24-bits per pixel without a palette and in a run-length encoded format.

This documentation describes three types of DIB files:

- Windows version 3.0 device-independent bitmap files
- OS/2 Presentation Manager version 1.2 device-independent bitmap files
- RIFF device-independent bitmap files

The Windows 3.0 and Presentation Manager 1.2 DIBs are similar, so they are discussed together.

Overview of DIB Structure

Windows 3.0 and Presentation Manager 1.2 DIB files consist of the following sequence of data structures:

- A file header
- Bitmap information header
- Color table
- Array of bytes that defines the bitmap bits

The following sections describe each of these structures.

Bitmap File Header

The bitmap file header contains information about the type, size, and layout of a device-independent bitmap (DIB) file. In both the Windows 3.0 and Presentation Manager 1.2 DIBs, it is defined as a BITMAPFILEHEADER data structure:

The following code illustrates how to define a bitmap file header.

```
typedef struct tagBITMAPFILEHEADER {
    USHORT    bfType;
    ULONG     bfSize;
    USHORT    bfReserved1;
    USHORT    bfReserved2;
    ULONG     bfOffBits;
} BITMAPFILEHEADER;
```

The BITMAPFILEHEADER data structure contains the following fields.

Field	Description
<i>bfType</i>	Specifies the file type. It must consist of the character sequence BM (USHORT value 0x4D42).
<i>bfSize</i>	Specifies the file size in bytes.
<i>bfReserved1</i>	Reserved. Must be set to 0.
<i>bfReserved2</i>	Reserved. Must be set to 0.
<i>bfOffBits</i>	Specifies the byte offset from the BITMAPFILEHEADER structure to the actual bitmap data in the file.

Bitmap Information Header

The BITMAPINFO and BITMAPCOREINFO data structures define the dimensions and color information for Windows 3.0 and Presentation Manager 1.2 DIBs, respectively. These structures are defined as follows:

Windows 3.0 DIB

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD bmiColors[1];
} BITMAPINFO;
```

Presentation Manager 1.2 DIB

```
typedef struct BITMAPCOREINFO {
    BITMAPCOREHEADER  bmciHeader;
    RGBTRIPLE  bmciColors[1];
} BITMAPCOREINFO;
```

These structures are alike essentially; this section describes both structures simultaneously. Each field name for the Windows BITMAPINFO structure is followed by the corresponding field name for the Presentation Manager BITMAPCOREINFO 1.2 structure, in parentheses. The following table describes these fields.

Windows (PM) Fields	Description
<i>bmiHeader</i> (<i>bmciHeader</i>)	Specifies information about the dimensions and color format of the DIB. The BITMAPINFOHEADER and BITMAPCOREHEADER data structures are described in the next section.
<i>bmiColors</i> (<i>bmciColors</i>)	Specifies the DIB color table. The RGBQUAD and RGBTRIPLE data structures are described in Bitmap Color Table .

Information Header Structures

The BITMAPINFOHEADER and BITMAPCOREHEADER structures contain information about the dimensions and color format of Windows 3.0 and Presentation Manager 1.2 DIBs, respectively. They are defined as follows:

Windows 3.0 DIB

```
typedef struct tagBITMAPINFOHEADER {
    ULONG  biSize;
    ULONG  biWidth;
    ULONG  biHeight;
    USHORT biPlanes;
```

```

USHORT biBitCount;
ULONG biCompression;
ULONG biSizeImage;
ULONG biXPelsPerMeter;
ULONG biYPelsPerMeter;
ULONG biClrUsed;
ULONG biClrImportant;
} BITMAPINFOHEADER;

```

Presentation Manager 1.2 DIB

```

typedef struct tagBITMAPCOREHEADER {
    ULONG bcSize;
    USHORT bcWidth;
    USHORT bcHeight;
    USHORT bcPlanes;
    USHORT bcBitCount;
} BITMAPCOREHEADER;

```

Again, because of the similarity of the Windows 3.0 and Presentation Manager structures, their common fields are described together. Each field name for the Windows structure is followed by the corresponding field name for the Presentation Manager structure, in parentheses.

Common Fields

The following fields are used in both the Windows 3.0 and Presentation Manager 1.2 formats:

Windows (PM) Field	Description
<i>biSize</i> (<i>bcSize</i>)	Specifies the number of bytes required by the BITMAPINFOHEADER structure. You can use this field to distinguish between Windows 3.0 and Presentation Manager 1.2 DIBs.
<i>biWidth</i> (<i>bcWidth</i>)	Specifies the width of the DIB in pixels.
<i>biHeight</i> (<i>bcHeight</i>)	Specifies the height of the DIB in pixels.
<i>biPlanes</i> (<i>bcPlanes</i>)	Specifies the number of planes for the target device. Must be set to 1.
<i>biBitCount</i> (<i>bcBitCount</i>)	Specifies the number of bits-per-pixel.

Windows Fields

The following fields are used only in the Windows 3.0 BITMAPINFOHEADER structure:

Field	Description						
<i>biCompression</i>	Specifies the type of compression for a compressed bitmap. It can be one of the following values: <table> <tr> <td>BI_RGB</td><td>Specifies that the bitmap is not compressed.</td></tr> <tr> <td>BI_RLE4</td><td>Specifies a run-length encoded format for bitmaps with 4 bits-per-pixel. The compression format is a 2-byte format consisting of a count byte followed by two word-length color indexes.</td></tr> <tr> <td>BI_RLE8</td><td>Specifies a run-length encoded format for bitmaps with 8 bits-per-pixel. The compression format is a 2-byte format consisting of a count byte followed by a color-index byte.</td></tr> </table> <p>See Windows 3.0 Bitmap Compression Formats for information about the encoding schemes.</p>	BI_RGB	Specifies that the bitmap is not compressed.	BI_RLE4	Specifies a run-length encoded format for bitmaps with 4 bits-per-pixel. The compression format is a 2-byte format consisting of a count byte followed by two word-length color indexes.	BI_RLE8	Specifies a run-length encoded format for bitmaps with 8 bits-per-pixel. The compression format is a 2-byte format consisting of a count byte followed by a color-index byte.
BI_RGB	Specifies that the bitmap is not compressed.						
BI_RLE4	Specifies a run-length encoded format for bitmaps with 4 bits-per-pixel. The compression format is a 2-byte format consisting of a count byte followed by two word-length color indexes.						
BI_RLE8	Specifies a run-length encoded format for bitmaps with 8 bits-per-pixel. The compression format is a 2-byte format consisting of a count byte followed by a color-index byte.						
<i>biSizeImage</i>	Specifies the size of the image (in bytes).						
<i>biXPelsPerMeter</i>	Specifies the horizontal resolution (in pixels per meter) of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that most closely matches the characteristics of the current device.						
<i>biYPelsPerMeter</i>	Specifies the vertical resolution (in pixels per meter) of the target device for the bitmap.						
<i>biClrUsed</i>	Specifies the number of color values in the color table actually used by the bitmap. Possible values follow.						

0	Bitmap uses the maximum number of colors corresponding to the value of the <i>usBitCount</i> field.
Nonzero	<p>If the <i>usBitCount</i> value is less than 24, the <i>biClrUsed</i> value indicates the actual number of colors that the graphics engine or device driver will access.</p> <p>If the <i>usBitCount</i> value is 24, the <i>biClrUsed</i> value indicates the size of the reference color table used to optimize performance of Windows color palettes.</p> <p>If the bitmap is <i>packed</i> (that is, a bitmap in which the bitmap array immediately follows the BITMAPINFO header and which is referenced by a single pointer), the <i>biClrUsed</i> field must be set to 0 or to the actual size of the color table. Interpreting the Color Table for more information on how this field affects the interpretation of the color table.</p>
<i>biClrImportant</i>	Specifies the number of color indexes that are considered important for displaying the bitmap. If this value is 0, then all colors are important.

Bitmap Color Table

The color table is a collection of 24-bit RGB values. There are as many entries in the color table as there are colors in the bitmap. The color table is not available for bitmaps with 24 color bits because each pixel is represented by 24-bit RGB values in the actual bitmap data area.

Color Table Structure

The color table for Windows 3.0 and Presentation Manager 1.2 DIBs consists of an array of RGBQUAD and RGBTRIPLE structures, respectively. These structures are defined as follows:

Windows 3.0 DIB

```
typedef struct tagRGBQUAD {
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;
```

Presentation Manager 1.2 DIB

```
typedef struct tagRGBTRIPLE {
    BYTE rgbtBlue;
    BYTE rgbtGreen;
    BYTE rgbtRed;
} RGBTRIPLE;
```

These structures are essentially alike, so they are described together. Each field name for the Windows RGBQUAD structure is followed by the corresponding field name for the Presentation Manager RGBTRIPLE structure, in parentheses.

Order of Colors

The colors in the table should appear in order of importance. This can help a device driver render a bitmap on a device that cannot display

as many colors as there are in the bitmap. If the DIB is in Windows 3.0 format, the driver can use the *biClrImportant* field of the BITMAPINFOHEADER structure to determine which colors are important.

Field Descriptions

The RGBQUAD (RGBTRIPLE) structure contains the following fields:

Windows (PM) Field	Description
<i>rgbBlue</i> (<i>rgbtBlue</i>)	Specifies the blue intensity.
<i>rgbGreen</i> (<i>rgbtGreen</i>)	Specifies the green intensity.
<i>rgbRed</i> (<i>rgbtRed</i>)	Specifies the red intensity.
<i>rgbReserved</i> (no PM equivalent)	Not used. Must be set to 0.

Locating the Color Table

An application can use the *biSize* (*bcSize*) field of the BITMAPINFOHEADER (BITMAPCOREHEADER) structure to locate the color table. Each of the following statements assigns the *pColor* variable the byte offset of the color table from the beginning of the file:

```
/* Windows 3.0 DIB */
pColor = (LPSTR)pBitmapInfo + (USHORT)pBitmapInfo->biSize
/* Presentation Manager 1.2 DIB */
pColor = (LPSTR)pBitmapCoreInfo + (USHORT)pBitmapCoreInfo->bcSize
```

Interpreting the Color Table

The *biSize* (*bcSize*) field of the BITMAPINFOHEADER (BITMAPCOREHEADER) structure specifies how many bits define each pixel and specifies the maximum number of colors in the bitmap. Its value affects your interpretation of the color table.

The *biSize* (*bcSize*) field can have any of the following values:

Value	Meaning
1	The bitmap is monochrome, and the color table contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the color table. If the bit is set, the pixel has the color of the second entry in the table.
4	The bitmap has a maximum of 16 colors. Each pixel in the bitmap is represented by a 4-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, then the byte represents 2 pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the 16th table entry.
8	The bitmap has a maximum of 256 colors. Each pixel in the bitmap is represented by a byte-sized index into the color table. For example, if the first byte in the bitmap is 0x1F, then the first pixel has the color of the thirty-second table entry.
24	The bitmap has a maximum of 224 colors. The <i>bmiColors</i> (<i>bmcColors</i>) field is NULL, and each three bytes in the bitmap array represent the relative intensities of red, green, and blue, respectively, of a pixel.

Note on Windows DIBs
For Windows 3.0 DIBs, the field of the BITMAPINFOHEADER structure specifies the number of color indexes in the color table actually used by the bitmap. If the *biClrUsed* field is set to 0, the bit map uses the maximum number of colors corresponding to the value of the field.

Bitmap Data

The bits in the array are packed together, but each line of pixels, or scan line, must be zero-padded to end on a LONG boundary. When the bitmap is in memory, segment boundaries can appear anywhere in the bitmap. The origin of the bitmap is the lower-left corner. The following section describes compression formats for the Windows 3.0 bitmap data.

Windows 3.0 Bitmap Compression Formats

Windows supports run-length encoded formats for compressing 4- and 8-bit bitmaps. Compression reduces the disk and memory storage required for the bitmap. The following sections describe the compression formats.

Compression of 8-Bit-Per-Pixel DIBs

When the *biCompression* field is set to BI_RLE8, the bitmap is compressed using a run-length encoding format for an 8-bit bitmap. This format uses two modes:

- Encoded
- Absolute

Both modes can occur anywhere throughout a single bitmap.

Encoded Mode

Encoded mode consists of two bytes. The first byte specifies the number of consecutive pixels to be drawn using the color index contained in the second byte. Also, the first byte of the pair can be set to 0 to indicate an escape that denotes an end of line, end of bit map, or a delta. The interpretation of the escape depends on the value of the second byte of the pair. In encoded mode, the second byte has a value of 0 - 2.

The following table lists the meaning of the second byte values:

Second Byte	Meaning
0	End of line.
1	End of bitmap.
2	Delta. The two bytes following the escape contain unsigned values indicating the horizontal and vertical offset of the next pixel from the current position.

Absolute Mode

Absolute mode is signalled by the first byte set to 0 and the second byte set to a value between 03H and FFH. The second byte represents the number of bytes that follow, each of which contains the color index of a single pixel.

Each run must be aligned on a word boundary.

The following example shows the hexadecimal values of an 8-bit RLE bitmap. Under **Expanded Data**, the 2-digit values represent a color index for a single pixel.

Compressed Data	Expanded Data
03 04	04 04 04
05 06	06 06 06 06 06
00 03 45 56 67 00	45 56 67
02 78	78 78
00 02 05 01	move 5 right and 1 down
02 78	78 78
00 00	end of line
09 1E	1E 1E 1E 1E 1E 1E 1E 1E 1E
00 01	end of RLE bitmap

Compression of 4-Bit-Per-Pixel DIBs

When the field is set to BI_RLE4, the bitmap is compressed using a run-length encoding format for a 4-bit bitmap. This format uses two modes:

- Encoded mode
- Absolute mode

Encoded Mode

In encoded mode, the first byte of the pair contains the number of pixels to be drawn using the color indexes in the second byte.

The second byte contains two color indexes, one in its high-order nibble (that is, its high-order four bits) and one in its low-order nibble.

The first of the pixels is drawn using the color specified by the high-order nibble; the second, using the color in the low-order nibble; the third, with the color in the high-order nibble; and so on, until all the pixels specified by the first byte have been drawn.

Also, the first byte of the pair can be set to 0 to indicate an escape that denotes an end of line, end of bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair. In encoded mode, the second byte has a value from 00H to 02H.

Absolute Mode

In absolute mode, the first byte contains 0, the second byte contains the number of color indexes that follow, and subsequent bytes contain color indexes in their high- and low-order nibbles, one color index for each pixel.

Each run must be aligned on a word boundary.

The end-of-line, end-of-bitmap, and delta escapes valid for BI_RLE8 also apply to BI_RLE4.

The following example shows the hexadecimal values of a 4-bit RLE bitmap. Under **Expanded Data**, the 1-digit values represent a color index for a single pixel.

Compressed Data	Expanded Data
03 04	0 4 0
05 06	0 6 0 6 0
00 06 45 56 67 00	4 5 5 6 6 7
04 78	7 8 7 8
00 02 05 01	move 5 right and 1 down
04 78	7 8 7 8
00 00	end of line
09 1E	1 E 1 E 1 E 1 E 1
00 01	end of RLE bitmap

RIFF Device-Independent Bitmap File Format

There are two types of RIFF device-independent bitmap (RDIB) formats:

- A simple RDIB consisting of a DIB file enclosed in a RIFF chunk.
- An extended RDIB that allows the creation of more complex bitmaps.

To ensure that the maximum number of programs will accept an RDIB file, programs that adopt the extended RDIB format also must accept simple RDIB files. Both formats are described in the following sections.

Simple RDIB Format

The simple RDIB format consists of a Windows 3.0 or Presentation Manager 1.2 DIB enclosed in a *RIFF* chunk. Enclosing the DIB in a RIFF chunk permits the file to be consistently identified; for example, an INFO list can be included in the file.

The simple RDIB form is defined as follows, using the standard RIFF form definition notation:

```
<RDIB-form>      RIFF ( 'RDIB_data( <DIB-data> ) )
```

The **<DIB-data>** format is defined in [Device-Independent Bitmap File Format](#).

Extended RDIB Format

The extended RDIB format, designed to incorporate enhancements such as impression, is defined as follows:

```
<RDIB-form>
    RIFF( 'RDIB'
        <bmhd-ck>          /* Bitmap header chunk */
        [ <pal-file> |     /* Internal palette chunk */
          <XPAL-ck> ]      /* External palette chunk */
        <bitmap-data> )    /* Bitmap data */
```

The **<pal-file>** chunk can be any of the palette-file formats described in [Palette File Format](#). The **<bmhd-ck>**, **<XPAL-chunk>**, and **<bitmap-data>** are described in the following sections.

Bitmap Header Chunk

The **<bmhd-ck>** bitmap header chunk is defined as follows:

```
<bmhd-chunk>  bmhd( struct {
    ULONG      ulMemSize;      /* If ulPelFormat is 'data', only these */
    ULONG      ulPelFormat;    /* four fields are present. */
    USHORT     wTransType;
    ULONG      ulTransVal;
    ULONG      ulHdrSize;      /* Fields from ulHdrSize forward match */
    ULONG      ulWidth;        /* the Windows BITMAPINFOHEADER */
    ULONG      ulHeight;       /* structure, although some fields can */
    USHORT     ulPlanes;       /* contain new values. */
    USHORT     usBitCount;
    ULONG      ulCompression;
    ULONG      ulSizeImage;
    ULONG      ulXPelsPerMeter;
    ULONG      ulYPelsPerMeter;
    ULONG      ulClrUsed;
    ULONG      ulClrImportant;
} )
```

If the *ulCompression* field equals BI_RGB or BI_RLE8 or BI_RLE4, then the extended RDIB has the same bitmap format as a simple RDIB.

Each pixel format defines the orientation or position of the bitmap origin. Windows bitmaps (identified by a value of *data* in the *ulPelFormat* field) have the origin at the bottom left. By default, the other formats have their origin at the top left.

Field	Description				
<i>ulMemSize</i>	Equal to the size of the bitmap bits if the bits are uncompressed. For RDIBs with <i>ulPelFormat</i> equal to <i>data</i> , <i>ulMemSize</i> has one of the following values:				
	<table><tr><th>Image Type</th><th>Field Value</th></tr><tr><td>Non-RLE</td><td>Same as <i>ulSizeImage</i> value</td></tr></table>	Image Type	Field Value	Non-RLE	Same as <i>ulSizeImage</i> value
Image Type	Field Value				
Non-RLE	Same as <i>ulSizeImage</i> value				

8-bit RLE	Size as an uncompressed, 8-bit image
4-bit RLE	Size as an uncompressed, 4-bit image

ulPelFormat

Specifies a FOURCC code defining the pixel format of the bitmap data. The bitmap data is stored in a chunk (or chunks) that has the same chunk ID as contained in *ulPelFormat*. The compression scheme and pixel depth of the bitmap data are recorded in the *ulCompression* and *ulBitCount* fields. The current bitmap data values are as follows:

Value	Bitmap Data Location and Format
<i>data</i>	Bitmap data is stored in a <i>data</i> chunk using the format defined for Windows 3.0 device-independent bitmaps (DIBs). An application can display the bitmap properly even if the fields after (and including) <i>ulMemSize</i> are ignored.
<i>palb</i>	Bitmap data is stored in a <i>palb</i> chunk. The pixel format is one of the Windows 3.0 RGB palettized formats (1 to 8 bpp, depending on the value of the <i>ulBitCount</i> field).
<i>rgbb</i>	Bitmap data is stored in an <i>rgbb</i> chunk. Pixel format is packed, unpalettized RGB represented at 16, 24, or 32 bits per pixel. The following table shows the ordering of the RGB bits for each pixel-depth value. The first extra bit (if present) is the high-order bit.

<i>ulBitCount</i>	Extra	Red	Green
15	1	5	5
16	0	5	6
24	0	8	8
32	8	8	8

<i>yuvb</i>	Bitmap data is stored in a <i>yuvb</i> chunk. Pixel format is packed, unpalettized YUV. The exact pixel format is currently undefined. By the time this draft is final, the pixel format will be defined similarly to the <i>rgbb</i> definition.
-------------	---

usTransType

Specifies the type of transparency representation, if any, used for this image. This is normally used for image overlay applications, where one image can be on top of another visually, and all pels of the transparency color must not be drawn. Examples include sprites, clip art, and motion video overlay. Wherever the transparency color occurs in the picture, the background must be visible.

This information is stored with the image, so that multiple images that use the same color map can all have different transparency color. There are five different values for the transparency variable. These are:

Value	Result
<i>BITT_NONE (0x0000)</i>	No pels are considered transparent in this image.
<i>BITT_MAPINDEX (0x0001)</i>	One of the color map/palette entries must be considered the transparency color. All instances of this pel must not be drawn, and the existing background must be allowed to show through.
<i>BITT_SINGLECOLOR (0x0002)</i>	A single RGB or YUV value is considered transparent and must not be drawn.
<i>BITT_BITPLANE (0x0003)</i>	An individual bit plane is considered

transparent, and all pels that have that bit or bits *on* must not be drawn.

BITT_MULTILEVEL
(0x0004)

A set of bits indicate multiple levels of transparency or opacity. This is usually used with 32-bit RGB, where the high 8 bits indicate transparency.

ulTransVal

These bytes allow the image definition to indicate the exact information about the transparent color. The information is dependent on the value of the **wTransType** as follows:

wTransType	ulTransVal Contents
BITT_NONE	Not used.
BITT_MAPINDEX	Specifies a palette index, either 0 through 16 or 0 through 255, depending on the number of palette entries.
BITT_SINGLECOLOR	Specifies an RGB or YUV value (2 to 4 bytes in size, depending on the pixel format specified by ulPelFormat). All pels that match ulTransVal should be considered transparent.
BITT_BITPLANE	Specifies a bit mask identifying the bits used to indicate a transparent pel. Any pel that has this set of bits set is totally transparent. This allows multiple colors to be considered transparent. This method works for palettized images; in this case, the value refers to a map entry that is considered transparent.
BITT_MULTILEVEL	<p>Specifies bits to use for transparency levels. These bits act as a mask on every pel, and each pel can be matched to the mask to determine the transparency level for the pel.</p> <p>For example, if ulTransVal has value 0xFF000000, then there are 256</p>

levels of transparency. Each pel can be evaluated against the mask. If the pel has a value FFxxxxxx, then it is fully transparent. If the pel has a value 00xxxxxx, then it is fully visible. If the pel has a value 7Fxxxxxx, the the pel is half visible.

ulHdrSize Specifies the size of the data portion of the **<bmhdr>** chunk. This is always 40, the size of the BITMAPINFOHEADER structure.

ulWidth Specifies the width of the DIB in pixels.

ulHeight Specifies the height of the DIB in pixels.

usPlanes Specifies the number of planes. This value is normally 1, but it can be 3 or 4 for 24-bit RGB and 32-bit RGB images, respectively. In a multiplane DIB, each color component (for example, red, green, and blue) is stored as a separate plane, and each plan is stored in a separate bitmap data chunk. For example, in a 3-plane, 24-bit 'rgbb' bitmap, the red colors are stored in one 'rgbb' chunk, the green colors in a second 'rgbb' chunk, and the blue colors in a third 'rgbb' chunk.

Allowing the separate RGB planes to be compressed independently can dramatically improve the compression ratio. The *usPlanes* value must be 1 if *ulPixelFormat* equals 'data'.

usBitCount Specifies the number of bits per pixel. If the *ulPixelFormat* field equals 'data', this field must contain values compatible with the Windows 3.0 DIB definition.

ulCompression Specifies the type of compression for a compressed bitmap. It can be one of the following values:

Value	Meaning
BI_NONE (0xFFFF0000)	Specifies that the bitmap is not compressed. Pixel values are <i>not</i> padded to four-byte boundaries.
BI_RGB (0x00000000)	Specifies that the bitmap is an uncompressed, 1-, 4-, 8-, or a 24-bit image. For 24-bit images, the palette is optional. Bitmap bits are represented as defined by Windows 3.0 for BI_RGB DIBs. The ulPixelFormat field must be set to 'data'.
BI_RLE8 (0x00000001)	Specifies a run-length encoded, compressed bitmap (as defined by Windows 3.0 BI_RLE8 DIBs). The palette is required. The ulPixelFormat field must be set to 'data'.
BI_RLE4 (0x00000002)	Specifies a run-length encoded, compressed bitmap (as defined by Windows 3.0 BI_RLE4 DIBs). The palette is required. The ulPixelFormat field must be set to 'data'.
BI_PACK (0xFF0001)	Specifies a simple PACKBITS byte compression scheme consisting of one-byte counts followed by byte data, in

the form:

<count byte *n*><data byte1><data byte2>
<data byte *n*>
<count byte *n*><data byte to repeat>

The high-order bit of the count byte *n* is
a decision bit:

***n* Value**

n < 0x80

n > 0x80

D
a
t
a
R
e
p
r
e
s
e
n
t
a
t
i
o
n

A
r
u
n
o
f
n
+
1
n
o
n
-
r
e
p
e
a
t
i
n
g
b
y
t
e
s
f
o
l
l
o
w
s
.

D

n = 0x80

BI_TRANS (0xFFFF0002)	Specifies transitional compression, using a table of byte transitions or sequences. See Transitional Compression .
BI_CCC (0xFFFF0003)	Specifies CCC compression, a method involving encoding each 4-by-4 block of the image using two colors.
BI_JPEGN (0xFFFF0004)	To be defined later, when the ISO completes the official specification.

<i>ulSizeImage</i>	Specifies the size in bytes of the compressed image.
<i>ulXPelsPerMeter</i>	Specifies the horizontal resolution in pixels per meter of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device. This field is set to zero if unused.
<i>ulYPelsPerMeter</i>	Specifies the vertical resolution in pixels per meter of the target device for the bitmap. This field is set to zero if unused.
<i>ulClrUsed</i>	Specifies the number of palette entries actually used by the bitmap. Possible values follow.

	Value	Result
	0	Bitmap uses the maximum number of colors corresponding to the value of the <i>usBitCount</i> field.
	Non-zero	<p>If the <i>usBitCount</i> is less than 24, <i>uiClrUsed</i> specifies the actual number of colors which the graphics engine or device driver will access.</p> <p>If the <i>usBitCount</i> field is set to 24, <i>uiClrUsed</i> specifies the size of the reference color table used to optimize performance of Windows color palettes.</p>
<i>uiClrImportant</i>		Specifies the number of color indexes that are considered important for displaying the bitmap. If this value is 0, then all colors are important.

Transitional Compression

If the *uiCompression* field is set to BI_TRANS, the data is transitionally compressed using a table of byte transitions or sequences. Values in the data indicate a table position to start at, and the table provides continuing references to other table positions. Transitional compression applies only to eight-bit data, either from an eight-bit palettized image or from a multi-plane image in which each color component is represented in eight bits.

The table consists of up to 256 16-byte rows at the beginning of the data section of the object. Nibbles (half-bytes) in the data section indicate an offset into a table row, at which location is stored the actual byte value. The actual value then becomes the row applicable to the next data nibble. The transitional encoding scheme is described more fully in a separate IBM document.

In transitional compression, the data section is a two-part compound object having the following items:

- A transition table
- The compressed image data

The transition table consists of an integer indicating the table size in bytes and a table of 16-byte rows. The first byte in each row is a row number and the next 15 are transition values. Rows are in descending sequence. The image is compressed according to the following rules:

- Data is in nibbles (half-bytes) or in nibble-pairs (successive half-bytes which may cross a byte boundary).
- The first byte is a nibble-pair. It is the first byte of the image and also the first row number.
- Following a nibble-pair is a series of transition nibbles (1[-15] ended by a terminator (0). Each transition nibble indicates an offset in the current row at which the next byte in the image is found; this value is also the next row number.
- The terminator indicates that the next image byte is not in the table, but instead in the following nibble-pair. This value is also the next row number.
- If the picture has an odd number of nibbles (i.e., it ends in the first half of the last byte), an extra zero nibble is included.

Palette Chunk

A PLT chunk represents a color table and consists of a valid PAL file. The PAL file format is defined in [Palette File Format](#).

External Palette Chunk

Instead of a PLT chunk, an RDIB may contain an XPLT chunk, which indicates that the bitmap's palette is stored outside the bitmap. The

palette might be stored in a separate file or as a separate compound file element. The XPLT chunk indicates the name and location of the external palette chunk and is defined as follows:

```
<XPLT-chunk>    XPLT( <fccLocation:FOURCC> <szPaletteName::ZSTR> )
```

The **fccLocation** contains one of the following FOURCC values specifying the location of the external chunk:

fccLocation Value	Chunk Location
'full'	Palette is located in an external file, and the szPaletteName value specifies a complete file name with path.
'file'	Palette is located in an external file, and the szPaletteName value specifies a file name without path.
'elem'	Palette is located in the same compound file containing the DIB. The szPaletteName value specifies the name of the compound file element.

The **szPaletteName** consists of a null-terminated string (ZSTR) containing the name of the external chunk containing the palette.

Bitmap Data Chunk

The <bitmap-data> contains bitmap data in the format specified by the *biPeflFormat* field of the <bmhd-chunk>.

MIDI and RIFF MIDI File Formats

The Musical Instrument Digital Interface (MIDI) file format represents a Standard MIDI File, as defined by the MIDI Manufacturers Association. A MIDI file contains commands instructing instruments to play specific notes and perform other operations.

The specifications for MIDI and MIDI files can be obtained from the following organization:

International MIDI Association (IMA)
5316 W. 57th Street
Los Angeles, CA 90056

The 'RMID' format consists of a standard MIDI file enclosed in a RIFF chunk. Enclosing the MIDI file in a 'RIFF' chunk allows the file to be consistently identified; for example, an 'INFO' list can be included in the file.

The 'RMID' form is defined as follows, using the standard RIFF form definition:

```
<RMID-form>    RIFF ( 'RMID' data( <MIDI-data> ) )
```

The <MIDI-data> is equivalent to a Standard MIDI File.

Palette File Format

The Palette (PAL) File Format represents a logical palette, which is a collection of colors represented as RGB values. There are two types of PAL formats:

- A simple PAL format
- An extended PAL format

Simple PAL Format

The simple PAL format is defined as follows:

```
RIFF('PAL' data( <palette:LOGPALETTE> ))
```

LOGPALETTE is the Windows 3.0 logical palette structure, defined as follows:

```
typedef struct tagLOGPALETTE {
    USHORT      palVersion;
    USHORT      palNumEntries;
    PALETTEENTRY palPalEntry[];
} LOGPALETTE;
```

The LOGPALETTE structure fields are as follows:

Field	Description
<i>palVersion</i>	Specifies the Windows version number for the structure.
<i>palNumEntries</i>	Specifies the number of palette color entries.
<i>palPalEntry[]</i>	Specifies an array of PALETTEENTRY data structures that define the color and usage of each entry in the logical palette.

The colors in the palette entry table should appear in order of importance. This is because entries earlier in the logical palette are most likely to be placed in the system palette.

The PALETTEENTRY data structure specifies the color and usage of an entry in a logical color palette. The structure is defined as follows:

```
typedef struct tagPALETTEENTRY {
    BYTE      peRed;
    BYTE      peGreen;
    BYTE      peBlue;
    BYTE      peFlags;
} PALETTEENTRY;
```

The PALETTEENTRY structure fields are as follows:

Field	Description
<i>peRed</i>	Specifies the intensity of red for the palette entry color.
<i>peGreen</i>	Specifies the intensity of green for the palette entry color.
<i>peBlue</i>	Specifies the intensity of blue for the palette entry color.
<i>peFlags</i>	Specifies how the palette entry is to be used.

Extended PAL Format

The extended PAL format includes the following:

- A palette-header chunk
- A data chunk containing an RGB palette (consisting of a LOGPALETTE structure) or some other palette type, including YUV and XYZ palettes.

For an RGB palette, the extended PAL format is represented as follows:

```
RIFF('PAL' plth( <palette-header> ) data( <LOGPALETTE-data> ))
```

For a YUV palette, the extended PAL format is represented as follows:

```
RIFF('PAL' plth( <palette-header> ) yuvp( <YUV-LOGPALETTE-data> ))
```

Both the **<LOGPALETTE-data>** and **<YUV-LOGPALETTE-data>** use the Windows 3.0 LOGPALETTE structure, described in "Simple PAL Format," earlier in this section. The **<YUV-LOGPALETTE-data>** contains YUV values instead of RGB valves.

The "plth" chunk is defined as follows:

```
<plth-ck>  PLT( struct {
    ULONG  ulMapType;
    USHORT usWhite;      /* Fields from this point on are */
    USHORT usBlack;      /* optional. If they are included */
    USHORT usBorder;     /* but not used, set them to 0xFFFF. */
    USHORT usRegisteredMap;
    USHORT usCustomBase; /* If an application encounters a */
    USHORT usCustomCnt;  /* 'PLT' chunk smaller than shown */
    USHORT usRsvBase;    /* here, it should treat the missing */
    USHORT usRsvCount;   /* fields as unused. */
    USHORT usArtBase;
    USHORT usArtCnt;
    USHORT usNumIntense;
} )
```

The structure fields are described in the following:

Field	Description								
<i>ulMapType</i>	FOURCC code specifying the type of palette. Currently, the following palette types are identified: <table><tr><th>Code</th><th>Description</th></tr><tr><td>'data'</td><td>Specifies an RGB palette. Data chunk contains a LOGPALETTE structure.</td></tr><tr><td>'yuvp'</td><td>Specifies a YUV palette. Data chunk contains a YUV palette.</td></tr><tr><td>'xyzp'</td><td>Specifies an XYZ palette. Data chunk contains a XYZ palette.</td></tr></table>	Code	Description	'data'	Specifies an RGB palette. Data chunk contains a LOGPALETTE structure.	'yuvp'	Specifies a YUV palette. Data chunk contains a YUV palette.	'xyzp'	Specifies an XYZ palette. Data chunk contains a XYZ palette.
Code	Description								
'data'	Specifies an RGB palette. Data chunk contains a LOGPALETTE structure.								
'yuvp'	Specifies a YUV palette. Data chunk contains a YUV palette.								
'xyzp'	Specifies an XYZ palette. Data chunk contains a XYZ palette.								
<i>usWhite , usBlack</i>	Specify palette-map indices corresponding to the closest value of white and black. These identify the pair of colors with the best contrast for use in cursors, calibration, etc. These values are usually changed if the palette changes. Ignore these fields if they contain 0xFFFF.								
<i>usBorder</i>	Specifies the index of the palette entry to be used for any display-border regions, if supported by the display device. Ignore this field if it contains 0xFFFF.								
<i>usRegisteredMap</i>	<p>Specifies how many palette entries correspond to a registered color map. Registered entries are stored at the front of the palette. Ignore this field if it contains 0xFFFF.</p> <p>Registered map entries are always stored at the beginning of the palette, so <i>usRegisteredMap</i> also indicates the index of the first custom color in the palette. Registered color maps include predefined palettes for general use, forest/nature, or seashores. Currently defined values are the following:</p> <table><tr><td>PAL_UNREGISTERED (0xFFFF)</td><td>Color map does not contain colors from a registered color map.</td></tr><tr><td>PAL_VGA (0x0000)</td><td>Color map contains the standard 16 VGA colors.</td></tr><tr><td>PAL_AVC198 (0x0001)</td><td>Standard AVC 198-entry palette.</td></tr></table>	PAL_UNREGISTERED (0xFFFF)	Color map does not contain colors from a registered color map.	PAL_VGA (0x0000)	Color map contains the standard 16 VGA colors.	PAL_AVC198 (0x0001)	Standard AVC 198-entry palette.		
PAL_UNREGISTERED (0xFFFF)	Color map does not contain colors from a registered color map.								
PAL_VGA (0x0000)	Color map contains the standard 16 VGA colors.								
PAL_AVC198 (0x0001)	Standard AVC 198-entry palette.								
<i>usCustomBase</i>	Specifies the index of the first custom color of the palette. The beginning of the palette contains the entries of the registered map, so <i>usCustomBase</i> also indicates the number of entries in the registered palette. Map entries starting with <i>usCustomBase</i> comprise additional								

	custom colors used in the bitmap. Ignore this value if <i>usRegisteredPalette</i> is PAL_UNREGISTERED, or if <i>usCustomBase</i> contains 0xFFFF.
<i>usCustomCnt</i>	Specifies the number of custom colors in the palette. Ignore this value if <i>usRegisteredPalette</i> is PAL_UNREGISTERED, or if this field contains 0xFFFF.
<i>usRsvBase</i>	Specifies the index of the first reserved color of the palette. Reserved colors are those reserved for menus, text, and other screen elements. Reserved colors must be stored contiguously. Ignore this field if it contains 0xFFFF.
<i>usRsvCnt</i>	Specifies the number of reserved entries. Ignore this field if it contains 0xFFFF.
<i>usArtBase</i>	Specifies the index of the first art color of the palette. Art colors are colors used for text and drawing. Art colors consist of a number of hues, each of which has multiple intensities. The various intensities are used for anti-aliasing, a method of using different shades of a color to improve the quality of images displayed on low-resolution devices. For example, if the first art color is red anti-aliased to black with three intensities, the first three entries in the palette would be dark red, medium red, and bright red. The art colors constitute an array, and all hues have the same number of intensities. The user can set both the number of hues and the number of intensities. Ignore these fields if they contain 0xFFFF.
<i>usArtCnt</i>	Specifies the number of art colors. Ignore this field if it contains 0xFFFF.
<i>usNumIntense</i>	Specifies the number of palette entries reserved for the anti-aliased levels of a given art color. This field must be present if <i>usArtBase</i> is present. Ignore this field if it contains 0xFFFF.

Rich Text Format (RTF)

The Rich Text Format (RTF) is a standard method of encoding formatted text and graphics using only 7-bit ASCII characters. Formatting includes different font sizes, faces, and styles, as well as paragraph alignment, justification, and tab control.

RTF is described in the *Microsoft Word Technical Reference: For Windows and OS/2*, published by Microsoft Press.

Waveform Audio File Format (WAVE)

This section describes the Waveform format, which represents digitized sound.

The WAVE form is defined as follows. Programs must expect (and ignore) any unknown chunks encountered, as with all RIFF forms. However, **<fmt-ck>** must always occur before **<wave-data>**, and both of these chunks are mandatory in a WAVE file:

```
<WAVE-form>
    RIFF( 'WAVE'
        <fmt-ck>                /* Format
        [<fact-ck>]             /* Fact chunk */
        [<cue-ck>]              /* Cue points */
        [<playlist-ck>]         /* Playlist */
        [<assoc-data-list>]      /* Associated data list */
        <wave-data> )          /* Wave data */
```

The WAVE chunks are described in the following sections.

WAVE Format Chunk

The WAVE format chunk **<fmt-ck>** specifies the format of the **<wave-data>**. The **<fmt-ck>** is defined as follows:

```
<fmt-ck>          fmt(      <common-fields>
                           <format-specific-fields> )
<common-fields>
    struct
    {
        USHORT  usFormatTag;          /* Format category          */
        USHORT  usChannels;           /* Number of channels       */
        ULONG   ulSamplesPerSec;      /* Sampling rate            */
        ULONG   ulAvgBytesPerSec;     /* For buffer estimation    */
        USHORT  usBlockAlign;         /* Data block size         */
    }
```

The fields in the **<common-fields>** chunk are as follows:

Field	Description
<i>usFormatTag</i>	<p>A number indicating the WAVE format category of the file. The content of the <format-specific-fields> portion of the 'fmt' chunk, and the interpretation of the waveform data, depend on this value.</p> <p>You must register any new WAVE format categories. See Registering Multimedia Formats for information on registering WAVE format categories.</p> <p>WAVE Format Categories lists the currently defined WAVE format categories.</p>
<i>usChannels</i>	The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo.
<i>ulSamplesPerSec</i>	The sampling rate (in samples per second) at which each channel should be played.
<i>ulAvgBytesPerSec</i>	The average number of bytes per second at which the waveform data should be transferred. Playback software can estimate the buffer size using this value.
<i>usBlockAlign</i>	The block alignment (in bytes) of the waveform data. Playback software needs to process a multiple of <i>usBlockAlign</i> bytes of data at a time, so the value of <i>usBlockAlign</i> can be used for buffer alignment.

The **<format-specific-fields>** consist of zero or more bytes of parameters. Which parameters occur depends on the WAVE format category-see the following section for details. Playback software should be written to allow for (and ignore) any unknown **<format-specific-fields>** parameters that occur at the end of this field.

WAVE Format Categories

The format category of a WAVE file is specified by the value of the *usFormatTag* field of the 'fmt' chunk. The representation of data in **<wave-data>**, and the content of the **<format-specific-fields>** of the 'fmt' chunk, depend on the format category.

The currently defined open non-proprietary WAVE format categories are as follows:

usFormatTag Value	Format Category
WAVE_FORMAT_PCM (0x0001)	Pulse Code Modulation (PCM) format

The following are the registered proprietary WAVE format categories:

usFormatTag Value	Format Category
IBM_FORMAT_ADPCM (0x0103)	IBM AVC Adaptive Differential Pulse Code Modulation format

The following sections describe the Microsoft WAVE_FORMAT_PCM format.

Pulse Code Modulation (PCM) Format

If the *usFormatTag* field of the **<fmt-ck>** is set to WAVE_FORMAT_PCM, then the waveform data consists of samples represented in pulse code modulation (PCM) format. For PCM waveform data, the **<format-specific-fields>** is defined as follows:

```
<PCM-format-specific>
    struct
    {
        USHORT    usBitsPerSample;        /* Sample size */
    }
```

The *usBitsPerSample* field specifies the number of bits of data used to represent each sample of each channel. If there are multiple channels, the sample size is the same for each channel.

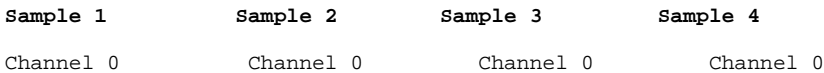
For PCM data, the *usAvgBytesPerSec* field of the 'fmt' chunk should be equal to the following formula rounded up to the next whole number:
usChannels x usBitsPerSample x ulSamplesPerSec x .125

The *usBlockAlign* field should be equal to the following formula, rounded to the next whole number:
usChannels x usBitsPerSample x .125

Data Packing for PCM WAVE Files

In a single-channel WAVE file, samples are stored consecutively. For stereo WAVE files, channel 0 represents the left channel, and channel 1 represents the right channel. The speaker position mapping for more than two channels is currently undefined. In multiple-channel WAVE files, samples are interleaved.

The following diagrams show the data packing for a 8-bit mono and stereo WAVE files:

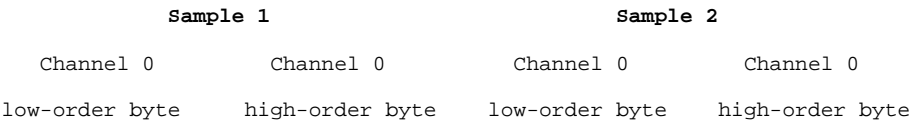


Data Packing for 8-Bit Mono PCM

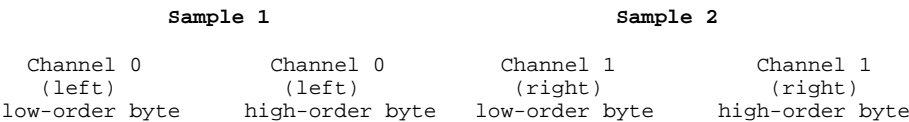


Data Packing for 8-Bit Stereo PCM

The following diagrams show the data packing for 16-bit mono and stereo WAVE files:



Data Packing for 16-Bit Mono PCM



Data Packing for 16-Bit Stereo PCM

Data Format of the Samples

Each sample is contained in an integer *i*. The size of *i* is the smallest number of bytes required to contain the specified sample size. The

least significant byte is stored first. The bits that represent the sample amplitude are stored in the most significant bits of i , and the remaining bits are set to zero.

For example, if the sample size (recorded in *nBitsPerSample*) is 12 bits, then each sample is stored in a two-byte integer. The least significant four bits of the first (least significant) byte is set to zero.

The data format and maximum and minimum values for PCM waveform samples of various sizes are as follows:

Sample Size	Data Format	Maximum Value	Minimum Value
One to eight bits	Unsigned integer	255 (0xFF)	0
Nine or more bits	Signed integer i	Largest positive value of i	Most negative value of i

For example, the maximum, minimum, and midpoint values for 8-bit and 16-bit PCM waveform data are as follows:

Format	Maximum Value	Minimum Value	Midpoint Value
8-bit PCM	255 (0xFF)	0	128 (0x80)
16-bit PCM	32767 (0x7FFF)	-32768 (-0x8000)	0

Examples of PCM WAVE Files

Example of a PCM WAVE file with 11.025 kHz sampling rate, mono, 8 bits per sample:

```
RIFF(      'WAVE'      fmt(1, 1, 11025, 11025, 1, 8)
                                data( <wave-data> ) )
```

Example of a PCM WAVE file with 22.05 kHz sampling rate, stereo, 8 bits per sample:

```
RIFF(      'WAVE'      fmt(1, 2, 22050, 44100, 2, 8)
                        data( <wave-data> ) )
```

Example of a PCM WAVE file with 44.1 kHz sampling rate, mono, 20 bits per sample:

```
RIFF(      'WAVE'      INFO(INAM("O Canada"Z))
                        fmt(1, 1, 44100, 132300, 3, 20)
                        data( <wave-data> ) )
```

Storage of WAVE Data

The **<wave-data>** contains the waveform data. It is defined as follows:

[illegible]

Note: The 'slnt' chunk represents silence, not necessarily a repeated zero volume or baseline sample. In 16-bit PCM data, if the last sample value played before the silence section is a 10000, then if data is still output to the D to A converter, it must maintain the 10000 value. If a zero value is used, a click may be heard at the start and end of the silence section. If play begins at a silence section, then a zero value might be used since no other information is available. A click might be created if the data following the silent section starts with a nonzero value.

FACT Chunk

The **<fact-ck>** fact chunk stores important information about the contents of the WAVE file. This chunk is defined as follows:

```
<fact-ck>      fact( <ulFileSize:ULONG> ) /* Number of samples */
```

The "fact" chunk is required if the waveform data is contained in a 'wavl' LIST chunk and for all compressed audio formats. The chunk is not required for PCM files using the 'data' chunk format.

The "fact" chunk will be expanded to include any other information required by future WAVE formats. Added fields will appear following the **<ulFileSize>** field. Applications can use the chunk size field to determine which fields are present.

Cue-Points Chunk

The **<cue-ck>** cue-points chunk identifies a series of positions in the waveform data stream. The **<cue-ck>** is defined as follows:

```
<cue-ck>      cue(      <ulCuePoints:ULONG> /* Count of cue points */
                        <cue-point>... )    /* Cue-point table */
<cue-point>    struct {
                        ULONG  ulName;
                        ULONG  ulPosition;
                        FOURCC  fccChunk;
                        ULONG  ulChunkStart;
                        ULONG  ulBlockStart;
                        ULONG  ulSampleOffset;
                    }
```

The **<cue-point>** fields are as follows:

Field	Description
<i>ulName</i>	Specifies the cue point name. Each <cue-point> record must have a unique <i>ulName</i> field.
<i>ulPosition</i>	Specifies the sample position of the cue point. This is the sequential sample number within the play order. See Playlist Chunk later in this document, for a discussion of the play order.
<i>fccChunk</i>	Specifies the name or chunk ID of the chunk containing the cue point.
<i>ulChunkStart</i>	Specifies the file position of the start of the chunk containing the cue point. This is a byte offset relative to the start of the data section of the 'wavl' LIST chunk.
<i>ulBlockStart</i>	Specifies the file position of the start of the block containing the position. This is a byte offset relative to the start of the data section of the 'wavl' LIST chunk.
<i>ulSampleOffset</i>	Specifies the sample offset of the cue point relative to the start of the block.

Examples of File Position Values

The following table describes the **<cue-point>** field values for a WAVE file containing multiple 'data' and 'slnt' chunks enclosed in a 'wavl'

LIST chunk:

Cue Point Location	Field	Value
In a 'slnt' chunk	<i>fccChunk</i>	FOURCC value 'slnt'.
	<i>ulChunkStart</i>	File position of the 'slnt' chunk relative to the start of the data section in the 'wavl' LIST chunk.
	<i>ulBlockStart</i>	File position of the data section of the 'slnt' chunk relative to the start of the data section of the 'wavl' LIST chunk.
	<i>ulSampleOffset</i>	Sample position of the cue point relative to the start of the 'slnt' chunk.
In a PCM 'data' chunk	<i>fccChunk</i>	FOURCC value 'data'.
	<i>ulChunkStart</i>	File position of the 'data' chunk relative to the start of the data section in the 'wavl' LIST chunk.
	<i>ulBlockStart</i>	File position of the cue point relative to the start of the data section of the 'wavl' LIST chunk.
	<i>ulSampleOffset</i>	Zero value.
In a compressed 'data' chunk	<i>fccChunk</i>	FOURCC value 'data'.
	<i>ulChunkStart</i>	File position of the start of the 'data' chunk relative to the start of the data section of the 'wavl' LIST chunk.
	<i>ulBlockStart</i>	File position of the enclosing block relative to the start of the data section of the 'wavl' LIST chunk. The software can begin the decompression at this point.
	<i>ulSampleOffset</i>	Sample position of the cue point relative to the start of the block.

The following table describes the <cue-point> field values for a WAVE file containing a single 'data' chunk:

Cue Point Location	Field	Value
Within PCM data	<i>fccChunk</i>	FOURCC value 'data'.
	<i>ulChunkStart</i>	Zero value.
	<i>ulBlockStart</i>	Zero value.
	<i>ulSampleOffset</i>	Sample position of the cue point relative to the start of the 'data' chunk.
In a compressed 'data' chunk	<i>fccChunk</i>	FOURCC value 'data'.
	<i>ulChunkStart</i>	Zero value.
	<i>ulBlockStart</i>	File position of the enclosing block relative to the start of the 'data' chunk. The software can begin the decompression at this point.
	<i>ulSampleOffset</i>	Sample position of the cue point relative to the start of the block.

Playlist Chunk

The **<playlist-ck>** playlist chunk specifies a play order for a series of cue points. The **<playlist-ck>** is defined as follows:

```

<playlist-ck>    plst(
                  <ulSegments:ULONG>    /* Count of play segments */
                  <play-segment>... )    /* Play-segment table */
<play-segment>  struct {
                  ULONG ulName;
                  ULONG ulLength;
                  ULONG ulLoops;
                }

```

The **<play-segment>** fields are as follows:

Field	Description
<i>ulName</i>	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
<i>ulLength</i>	Specifies the length of the section in samples.
<i>ulLoops</i>	Specifies the number of times to play the section.

Associated Data Chunk

The **<assoc-data-list>** associated data list provides the ability to attach information like labels to sections of the waveform data stream. The

<assoc-data-list> is defined as follows:

```
<assoc-data-list>      LIST(  'adtl'
                                <labl-ck>      /* Label                */
                                <note-ck>      /* Note                  */
                                <ltxt-ck>     /* Text with data length */
                                <file-ck>    ) /* Media file           */

<labl-ck>              labl(  <ulName:ULONG>
                                <data:ZSTR>  )

<note-ck>              note(  <ulName:ULONG>
                                <data:ZSTR>  )

<ltxt-ck>              ltxt(  <ulName:ULONG>
                                <ulSampleLength:ULONG>
                                <ulPurpose:ULONG>
                                <usCountry:USHORT>
                                <usLanguage:USHORT>
                                <usDialect:USHORT>
                                <usCodePage:USHORT>
                                <data:BYTE>... )

<file-ck>              file(  <usName:ULONG>
                                <ulwMedType:ULONG>
                                <fileData:BYTE>...)
```

Label and Note Information

The 'labl' and 'note' chunks have similar fields. The 'labl' chunk contains a label, or title, to associate with a cue point. The 'note' chunk contains comment text for a cue point. The fields are as follows:

Field	Description
<i>ulName</i>	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
<i>data</i>	Specifies a NULL-terminated string containing a text label (for the 'labl' chunk) or comment text (for the 'note' chunk).

Text with Data Length Information

The 'ltxt' chunk contains text that is associated with a data segment of specific length. The chunk fields are as follows:

Field	Description
<i>ulName</i>	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
<i>ulSampleLength</i>	Specifies the number of samples in the segment of waveform data.
<i>ulPurpose</i>	Specifies the type or purpose of the text. For example, <i>ulPurpose</i> can specify a FOURCC code like "scrp" for script text or "capt" for close-caption text.
<i>usCountry</i>	Specifies the country code for the text. See Country Codes for a current list of country codes.
<i>usLanguage</i> , <i>usDialect</i>	Specify the language and dialect codes for the text. See Language and Dialect Codes for a current list of language and dialect codes.
<i>usCodePage</i>	Specifies the code page for the text.

Embedded File Information

The 'file' chunk contains information described in other file formats (for example, an 'RDIB' file or an ASCII text file). The chunk fields are as follows:

Field	Description
<i>ulName</i>	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
<i>ulMedType</i>	Specifies the file type contained in the <i>fileData</i> field. If the <i>fileData</i> section contains a RIFF form, the <i>ulMedType</i> field is the same as the RIFF form type for the file. This field can contain a zero value.
<i>fileData</i>	Contains the media file.

RIFF Compound Files and Elements - Sharing and Access

The following tables show the relationship between sharing and accessing RIFF compound files and elements within those files, both within the current process and between processes. The following scenarios are presented:

- Element Opened Again - (any process)
- Elements Opened - (same process)
- Fully Qualified Element Open
- CF Opened Again - (same process)
- Element Open - (different process)
- CF Opened Again - (different process)

A indicates that the function is allowed. Both the access and sharing values must be *A* for the open to work. For example, in the CF Open - CF Open of the same process, if the first CF Open is RO - DW, then the next CF Open must be RO - DW. Any access other than RO fails. Any sharing other than DW also will fail.

Abbreviations of access modes are:

- RO - READ ONLY
- WO - WRITE ONLY
- WR - READ and WRITE

Abbreviations for sharing modes are:

- DR - DENY READ
- DW - DENY WRITE
- DN - DENY NONE
- EX - EXCLUSIVE (DENY ALL)

Element Opened Again							
Element Opened	Access			DR	Sharing		
	RO	WO	RW		DW	DN	EX
RO - DR	X	A	X	X	A	A	X

RO - DW	A	X	X	X	A	A	X
RO - DN	A	A	A	X	A	A	X
RO - EX	X	X	X	X	X	X	X
RW - DR	X	A	X	X	X	A	X
RW - DW	A	X	X	X	X	A	X
RW - DN	A	A	A	X	X	A	X
RW - EX	X	X	X	X	X	X	X
WO - DR	X	A	X	A	X	A	X
WO - DW	A	X	X	A	X	A	X
WO - DN	A	A	A	A	X	A	X
WO - EX	X	X	X	X	X	X	X

Any Process

Elements Opened

CF Opened	Access			Sharing			EX
	RO	WO	RW	DR	DW	DN	
RO - DR	A	X	X	A	A	A	A
RO - DW	A	X	X	A	A	A	A
RO - DN	A	X	X	A	A	A	A
RO - EX	A	X	X	A	A	A	A
RW - DR	A	A	A	A	A	A	A
RW - DW	A	A	A	A	A	A	A
RW - DN	A	A	A	A	A	A	A
RW - EX	A	A	A	A	A	A	A
WO - DR	X	A	X	A	A	A	A
WO - DW	X	A	X	A	A	A	A
WO - DN	X	A	X	A	A	A	A
WO - EX	X	A	X	A	A	A	A

Same Process

Fully Qualified Element Open

CF Opened	Access			DR	Sharing		EX
	RO	WO	RW		DW	DN	
RW - DN	A	A	A	A	A	A	A

CF Opened Again

CF Opened	Access			DR	Sharing		EX
	RO	WO	RW		DW	DN	
RO - DR	X	X	X	X	X	X	X
RO - DW	A	X	X	X	A	X	X
RO - DN	A	X	X	X	X	A	X
RO - EX	X	X	X	X	X	X	X
RW - DR	X	X	X	X	X	X	X
RW - DW	X	X	X	X	X	X	X
RW - DN	X	X	A	X	X	A	X
RW - EX	X	X	X	X	X	X	X
WO - DR	X	A	X	A	X	X	X
WO - DW	X	X	X	X	X	X	X
WO - DN	X	A	X	X	X	A	X
WO - EX	X	X	X	X	X	X	X

Same Process

Element Opened (different process)

CF Opened	Access			DR	Sharing		EX
	RO	WO	RW		DW	DN	
RO - DR	X	A	X	X	A	A	X
RO - DW	A	X	X	X	A	A	X
RO - DN	A	A	A	X	X	A	X
RO - EX	X	X	X	X	X	X	X

RW - DR	X	A	X	X	X	A	X
RW - DW	A	X	X	X	X	A	X
RW - DN	A	A	A	X	X	A	X
RW - EX	X	X	X	X	X	X	X
WO - DR	X	A	X	A	X	A	X
WO - DW	A	X	X	A	X	A	X
WO - DN	A	A	A	X	X	A	X
WO - EX	X	X	X	X	X	X	X

CF Opened Again

CF Opened	Access			DR	Sharing		EX
	RO	WO	RW		DW	DN	
RO - DR	X	A	X	X	A	A	X
RO - DW	A	X	X	X	A	A	X
RO - DN	A	A	A	X	A	A	X
RO - EX	X	X	X	X	X	X	X
RW - DR	X	A	X	X	X	A	X
RW - DW	A	X	X	X	X	A	X
RW - DN	A	A	A	X	X	A	X
RW - EX	X	X	X	X	X	X	X
WO - DR	X	A	X	A	X	A	X
WO - DW	A	X	X	A	X	A	X
WO - DN	A	A	A	A	X	A	X
WO - EX	X	X	X	X	X	X	X

Different Process

Return Codes

This section lists the following categories of OS/2 multimedia return codes including number, message, and a brief description:

- [Media Device Manager Return Codes](#)
- [Sync Stream Manager Return Codes](#)

- [MMIO Manager Return Codes](#)
- [DIVE Return Codes](#)

Media Device Manager Return Codes

- 5000 - MCIERR_SUCCESS
MMPM/2 command completed successfully.
- 5001 - MCIERR_INVALID_DEVICE_ID
Invalid device ID given.
- 5002 - MCIERR_NO_MASTER
Unable to find a Master.
- 5003 - MCIERR_UNRECOGNIZED_KEYWORD
Unrecognized keyword given in command string.
- 5004 - MCIERR_MASTER_CONFLICT
Only one device can be a Master.
- 5005 - MCIERR_UNRECOGNIZED_COMMAND
Unrecognized command.
- 5006 - MCIERR_HARDWARE
Hardware error.
- 5007 - MCIERR_INVALID_DEVICE_NAME
Invalid Device Name given.
- 5008 - MCIERR_OUT_OF_MEMORY
System out of memory.
- 5009 - MCIERR_DEVICE_OPEN
Device is already open.
- 5010 - MCIERR_CANNOT_LOAD_DRIVER
Cannot load MMPM/2 driver.
- 5011 - MCIERR_MISSING_COMMAND_STRING
Missing command string.
- 5012 - MCIERR_PARAM_OVERFLOW
Parameter overflow.
- 5013 - MCIERR_MISSING_STRING_ARGUMENT
Missing required string argument.
- 5014 - MCIERR_BAD_INTEGER
Bad integer given in command string.
- 5015 - MCIERR_PARSER_INTERNAL
Internal MMPM/2 parser error.
- 5016 - MCIERR_DRIVER_INTERNAL
Internal MMPM/2 driver error.
- 5017 - MCIERR_MISSING_PARAMETER
Missing parameter for this command.
- 5018 - MCIERR_UNSUPPORTED_FUNCTION
Function not supported by the media control driver being used.
- 5019 - MCIERR_FILE_NOT_FOUND
File not found.

5020 - MCIERR_DEVICE_NOT_READY
Device is not ready.

5021 - MCIERR_INTERNAL
MMPM/2 internal error.

5022 - MCIERR_DRIVER
Internal MMPM/2 driver error.

5023 - MCIERR_CANNOT_USE_ALL
Cannot use ALL keyword with this command.

5024 - MCIERR_MULTIPLE
Multiple defined.

5025 - MCIERR_EXTENSION_NOT_FOUND
Extension not found.

5026 - MCIERR_OUTOFRANGE
Value given is out of range.

5027 - MCIERR_CANNOT_ADD_ALIAS
Cannot add alias.

5028 - MCIERR_FLAGS_NOT_COMPATIBLE
Flags not compatible.

5029 - MCIERR_CANNOT_USE_NOUNLOAD
Cannot use NOUNLOAD flag with this command.

5030 - MCIERR_FILE_NOT_SAVED
File not saved.

5031 - MCIERR_DEVICE_TYPE_REQUIRED
Device type required.

5032 - MCIERR_DEVICE_LOCKED
Device is locked.

5033 - MCIERR_DUPLICATE_ALIAS
Duplicate alias.

5034 - MCIERR_INSTANCE_INACTIVE
Instance inactive.

5035 - MCIERR_COMMAND_TABLE
Command table error.

5037 - MCIERR_INI_FILE_LOCKED
The MMPM2.INI file is currently locked.

5040 - MCIERR_NO_AUDIO_SUPPORT
No audio support present.

5041 - MCIERR_NOT_IN_PM_SESSION
Not currently in a Presentation Manager Session.

5042 - MCIERR_DUPLICATE_KEYWORD
Duplicate keyword in command string.

5043 - MCIERR_COMMAND_STRING_OVERFLOW
Command string too long.

5044 - MCIERR_DRIVER_PROC_NOT_FOUND
MMPM/2 driver procedure address not found.

5045 - MCIERR_INVALID_DEVICE_TYPE
Invalid device type given.

5046 - MCIERR_INVALID_DEVICE_ORDINAL
Invalid device ordinal given.

5047 - MCIERR_HEADPHONES_NOT_SET

Headphones not set on.

5048 - MCIERR_SPEAKERS_NOT_SET
Speakers not set on.

5049 - MCIERR_SOUND_NOT_SET
Mute on.

5050 - MCIERR_INVALID_BUFFER
Invalid return buffer given.

5051 - MCIERR_INVALID_MEDIA_TYPE
Invalid media type given.

5052 - MCIERR_INVALID_CONNECTOR_INDEX
Invalid connector index.

5053 - MCIERR_NO_CONNECTION
No connection found.

5054 - MCIERR_INVALID_FLAG
Invalid flag specified for this command.

5055 - MCIERR_CANNOT_LOAD_DSP_MOD
DSP module not found.

5056 - MCIERR_ALREADY_CONNECTED
The connection already exists.

5057 - MCIERR_INVALID_CALLBACK_HANDLE
The window callback handle is invalid.

5058 - MCIERR_DRIVER_NOT_FOUND
MMPM2 driver not found.

5059 - MCIERR_DUPLICATE_DRIVER
Duplicate MMPM/2 driver found.

5060 - MCIERR_INI_FILE
MMPM2.INI file error.

5061 - MCIERR_INVALID_GROUP_ID
Invalid group ID given.

5062 - MCIERR_ID_ALREADY_IN_GROUP
Device ID already in group.

5063 - MCIERR_MEDIA_CHANGED
Media has been changed.

5064 - MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

5065 - MCIERR_UNSUPPORTED_FLAG
Flag not supported by this MMPM/2 driver for this command.

5066 - MCIERR_DRIVER_NOT_LOADED
MMPM/2 driver is not loaded.

5067 - MCIERR_INVALID_MODE
Device mode is invalid for this command.

5068 - MCIERR_INVALID_ITEM_FLAG
Invalid item flag specified for this command.

5069 - MCIERR_INVALID_TIME_FORMAT_FLAG
Invalid time format flag specified for this command.

5070 - MCIERR_SPEED_FORMAT_FLAG
Invalid speed format flag specified for this command.

5071 - MCIERR_INVALID_AUDIO_FLAG
Invalid audio flag specified for this command.

5072 - MCIERR_NODEFAULT_DEVICE
No default device specified.

5073 - MCIERR_DUPLICATE_EXTENSION
Duplicate Extension specified.

5074 - MCIERR_FILE_ATTRIBUTE
File Attribute error specified.

5075 - MCIERR_DUPLICATE_CUEPOINT
Duplicate cuepoint given.

5076 - MCIERR_INVALID_CUEPOINT
Invalid cuepoint given.

5077 - MCIERR_CUEPOINT_LIMIT_REACHED
Cue-point limit reached.

5078 - MCIERR_MISSING_ITEM
Item flag missing for this command.

5079 - MCIERR_MISSING_TIME_FORMAT
Time format flag missing for this command.

5080 - MCIERR_MISSING_SPEED_FORMAT
Speed format flag missing for this command.

5081 - MCIERR_INVALID_CONNECTOR_TYPE
Invalid connector type given.

5082 - MCIERR_TARGET_DEVICE_FULL
Target device is full.

5083 - MCIERR_UNSUPPORTED_CONN_TYPE
Connector type is not supported by this device.

5084 - MCIERR_CANNOT_MODIFY_CONNECTOR
Cannot enable or disable this connector.

5085 - MCIERR_RECORD_ABORTED
Record has been aborted.

5086 - MCIERR_GROUP_COMMAND
One or more devices in group failed command.

5087 - MCIERR_DEVICE_NOT_FOUND
Device cannot be found.

5088 - MCIERR_RESOURCE_NOT_AVAILABLE
Device resource is not available.

5089 - MCIERR_INVALID_IO_PROC
Invalid MMIO I/O procedure given.

5090 - MCIERR_WAVE_OUTPUTSINUSE
Output is in use.

5091 - MCIERR_WAVE_SETOUTPUTINUSE
Output is in use.

5092 - MCIERR_WAVE_INPUTSINUSE
Input is in use.

5093 - MCIERR_WAVE_SETINPUTINUSE
Input is in use.

5094 - MCIERR_WAVE_OUTPUTUNSPECIFIED
Output not specified.

5095 - MCIERR_WAVE_INPUTUNSPECIFIED
Input not specified.

5096 - MCIERR_WAVE_OUTPUTSUNSUITABLE
Output is not suitable.

5097 - MCIERR_WAVE_SETOUTPUTUNSUITABLE
Output is not suitable.

5098 - MCIERR_WAVE_INPUTSUNSUITABLE
Input is not suitable.

5099 - MCIERR_WAVE_SETINPUTUNSUITABLE
Input is not suitable.

5100 - MCIERR_SEQ_DIV_INCOMPATIBLE
Division format is not compatible with this device.

5101 - MCIERR_SEQ_PORT_INUSE
Port in use.

5102 - MCIERR_SEQ_PORT_NONEXISTENT
Port does not exist for this device.

5103 - MCIERR_SEQ_PORT_MAPNODEVICE
MIDI mapper device does not exist.

5104 - MCIERR_SEQ_PORT_MISCELLANEOUS
Port error.

5105 - MCIERR_SEQ_TIMER
MIDI timer error.

5106 - MCIERR_VDP_COMMANDCANCELLED
MMPM/2 command was cancelled by another MCI command.

5107 - MCIERR_VDP_COMMANDFAILURE
MMPM/2 command failed.

5108 - MCIERR_VDP_NOTSPUNUP
MMPM/2 command requires the videodisc player to be spun up.

5109 - MCIERR_VDP_NOCHAPTER
MMPM/2 command requires the videodisc to have chapters.

5110 - MCIERR_VDP_NOSIDE
Videodisc side cannot be determined.

5111 - MCIERR_VDP_NOSIZE
Videodisc size cannot be determined.

5112 - MCIERR_VDP_INVALID_TIMEFORMAT
MMPM/2 command does not support the time format.

5114 - MCIERR_CLIPBOARD_ERROR
A problem with the clipboard occurred.

5115 - MCIERR_CANNOT_CONVERT
Unable to convert clipboard format.

5116 - MCIERR_CANNOT_REDO
Cannot redo previous action.

5117 - MCIERR_CANNOT_UNDO
Cannot undo previous action.

5118 - MCIERR_CLIPBOARD_EMPTY
The clipboard is currently empty.

5119 - MCIERR_INVALID_WORKPATH
Work path given is not a valid OS/2 path.

5120 - MCIERR_INDETERMINATE_LENGTH
Cannot determine length.

5121 - MCIERR_DUPLICATE_EA

An Extended Attribute of this type already exists for another device.

- 5122 - MCIERR_INVALID_CONNECTION
This connection is not valid.
- 5123 - MCIERR_CHANNEL_OFF
Primary channel has been turned off.
- 5124 - MCIERR_CANNOT_CHANGE_CHANNEL
Can not change this channel.
- 5125 - MCIERR_FILE_IO
Error occurred during file read/write.
- 5126 - MCIERR_SYSTEM_FILE
Could not find VSH data for RTV record.
- 5127 - MCIERR_DISPLAY_RESOLUTION
Display resolution not supported by ActionMedia II adapter.
- 5128 - MCIERR_NO_ASYNC_PLAY_ACTIVE
Currently there is no asynchronous play active.
- 5129 - MCIERR_UNSUPP_FORMAT_TAG
Unsupported format tag.
- 5130 - MCIERR_UNSUPP_SAMPLESPERSEC
Unsupported sampling rate.
- 5131 - MCIERR_UNSUPP_BITSPERSAMPLE
Unsupported bits per sample.
- 5132 - MCIERR_UNSUPP_CHANNELS
Unsupported number of channels.
- 5133 - MCIERR_UNSUPP_FORMAT_MODE
Unsupported format mode.
- 5134 - MCIERR_NO_DEVICE_DRIVER
No device driver found.
- 5135 - MCIERR_CODEC_NOT_SUPPORTED
CODEC can not be found or cannot support the current video mode.
- 5136 - MCIERR_TUNER_NO_HW
The device does not have tuner capability.
- 5139 - MCIERR_TUNER_CHANNEL_SKIPPED
Channel skipped in region.
- 5140 - MCIERR_TUNER_CHANNEL_TOO_LOW
Channel too low for region.
- 5141 - MCIERR_TUNER_CHANNEL_TOO_HIGH
Channel too high for region.
- 5143 - MCIERR_TUNER_INVALID_REGION
The region file either does not exist or is invalid.
- 5144 - MCIERR_SIGNAL_INVALID
No valid signal present.
- 5144 - MCIERR_TUNER_MODE
Frequency was last set directly. Cannot determine region, channel, or fine tuning.
- 5146 - MCIERR_TUNER_REGION_NOT_SET
No region is defined.
- 5147 - MCIERR_TUNER_CHANNEL_NOT_SET
No channel is defined. Channel must be set whenever region is set.
- 5149 - MCIERR_UNSUPPORTED_ATTRIBUTE
Current mixer hardware does not support the attribute.

5256 - MCIERR_CUSTOM_DRIVER_BASE
Reserved for future use.

Sync Stream Manager Return Codes

5501 - ERROR_INVALID_STREAM
Invalid stream handle specified.

5502 - ERROR_INVALID_HID
Invalid handler ID specified.

5504 - ERROR_INVALID_OBJTYPE
Invalid object type specified.

5505 - ERROR_INVALID_FLAG
Invalid flag specified.

5506 - ERROR_INVALID_EVCB
Invalid Event Control Block.

5507 - ERROR_INVALID_EVENT
Invalid event type or handle specified.

5508 - ERROR_INVALID_MMTIME
Invalid MMTIME specified.

5509 - ERROR_INVALID_NUMSLAVES
Invalid number of slaves specified.

5510 - ERROR_INVALID_REQUEST
Invalid function requested.

5511 - ERROR_INVALID_SPCBKEY
Invalid Stream Protocol Key specified.

5512 - ERROR_INVALID_HNDLR_NAME
Invalid stream handler name specified.

5513 - ERROR_INVALID_PROTOCOL
Invalid Stream Protocol specified.

5514 - ERROR_INVALID_BUFFER_SIZE
Invalid buffer size specified.

5515 - ERROR_INVALID_BUFFER_RETURNED
Invalid buffer address returned.

5516 - ERROR_INVALID_ACB
Invalid Associate Control Block.

5517 - ERROR_INVALID_RECORD_RETURNED
Invalid record address returned.

5518 - ERROR_INVALID_MESSAGE
Invalid message

5599 - ERROR_STREAM_NOT_OWNER
Not owner of stream.

5600 - ERROR_STREAM_USED
Stream already used in sync group.

5601 - ERROR_STREAM_CREATION
Error during stream creation.

5602 - ERROR_STREAM_NOTMASTER
Stream is not master stream of sync group.

5603 - ERROR_STREAM_NOT_STOP
Stream is not stopped.

5604 - ERROR_STREAM_OPERATION
Error in stream operation.

5605 - ERROR_STREAM_STOP_PENDING
Stop stream is already in progress.

5606 - ERROR_STREAM_ALREADY_STOP
Stream has already been stopped.

5607 - ERROR_STREAM_ALREADY_PAUSE
Stream has already been paused.

5608 - ERROR_STREAM_NOT_STARTED
Stream has not been started.

5609 - ERROR_STREAM_NOT_ACTIVE
Stream is not active.

5610 - ERROR_START_STREAM
Error starting stream.

5611 - ERROR_MASTER_USED
Master stream used in another sync group.

5612 - ERROR_SPCBKEY_MISMATCH
SPCB key is mismatched.

5613 - ERROR_INSUFF_BUFFER
Insufficient buffer size specified.

5614 - ERROR_ALLOC_RESOURCES
Unable to allocate resources.

5615 - ERROR_ACCESS_OBJECT
Unable to access the object specified.

5616 - ERROR_HNDLR_REGISTERED
Stream Handler already registered.

5617 - ERROR_DATA_ITEM_NOT_SPECIFIED
Data item has not been specified.

5618 - ERROR_INVALID_SEQUENCE
Invalid sequence of events occurred.

5619 - ERROR_INITIALIZATION
Error during initialization.

5620 - ERROR_READING_INI
Error reading the SPI.INI file.

5621 - ERROR_LOADING_HNDLR
Error loading a Stream Handler.

5622 - ERROR_HNDLR_NOT_FOUND
Stream Handler not found in system.

5623 - ERROR_SPCB_NOT_FOUND
SPCB not found.

5624 - ERROR_DEVICE_NOT_FOUND
Device not found.

5625 - ERROR_TOO_MANY_EVENTS
Too many events for event queue.

5626 - ERROR_DEVICE_OVERRUN
Device overrun occurred.

5627 - ERROR_DEVICE_UNDERRUN
Device underrun occurred.

5628 - ERROR_HNDLR_NOT_IN_INI
Stream Handler not found in SPI.INI file.

5629 - ERROR_QUERY_STREAM_TIME
Time not available for this stream.

5630 - ERROR_DATA_ITEM_NOT_SEEKABLE
Data item cannot be seeked.

5631 - ERROR_NOT_SEEKABLE_BY_TIME
Not seekable by time.

5632 - ERROR_NOT_SEEKABLE_BY_BYTES
Not seekable by bytes.

5633 - ERROR_STREAM_NOT_SEEKABLE
Stream cannot be seeked.

5635 - ERROR_PLAYLIST_STACK_OVERFLOW
Playlist stack overflow.

5636 - ERROR_PLAYLIST_STACK_UNDERFLOW
Playlist stack underflow.

5637 - ERROR_LOCKING_BUFFER
Error locking down buffer.

5638 - ERROR_UNLOCKING_BUFFER
Error unlocking buffer.

5639 - ERROR_SEEK_PAST_END
Seek past end of object.

5640 - ERROR_SEEK_BACK_NOT_SUPPORTED
Seek back not supported.

5641 - ERROR_INTERNAL_ERROR
Internal memory error occurred.

5642 - ERROR_INTERNAL_CORRUPT
Internal memory pool corrupted.

5643 - ERROR_INSUFF_MEM
Insufficient memory.

5644 - ERROR_LARGE_SEEK_BY_TIME
Large seek by time.

5645 - ERROR_STREAM_PREROLLING
Function not allowed while prerolling.

5646 - ERROR_INI_FILE
Error encountered in SPI.INI file.

5647 - ERROR_SEEK_BEFORE_BEGINNING
Attempt to seek before beginning of object.

5648 - ERROR_TOO_MANY_HANDLERS
Stream Handler table full.

5649 - ERROR_ALLOC_HEAP
Error allocating memory from heap.

5650 - ERROR_END_OF_PLAYLIST
End of playlist encountered.

5651 - ERROR_TOO_MANY_STREAMS

Too many streams active.

5652 - ERROR_FILE_FORMAT_INCORRECT
File format incorrect.

5653 - ERROR_DESTROY_STREAM
Error destroying stream.

5654 - ERROR_INVALID_NUMMASTERS
Invalid number of streams in *ulNumMasters* of [SpiDetermineSyncMaster](#).

5655 - ERROR_MASTER_CONFLICT
Stream master conflict; two or more stream have master capabilities only.

5656 - ERROR_NO_MASTER
No stream can be a master stream.

5657 - ERROR_NO_SYNC
Stream cannot be in a synchronization group.

5900 - ERROR_BUFFER_NOT_AVAILABLE
Buffer is not available.

5901 - ERROR_TOO_MANY_BUFFERS
Too many buffers.

5902 - ERROR_TOO_MANY_RECORDS
Too many records.

MMIO Manager Return Codes

6501 - MMIOERR_UNBUFFERED
The specified file is not opened for buffered I/O.

6502 - MMIOERR_CANNOTWRITE
The buffer could not be written to disk.

6503 - MMIOERR_CHUNKNOTFOUND
The end of the file or parent chunk was reached.

6504 - MMIOERR_INVALID_HANDLE
The handle passed was not correct.

6505 - MMIOERR_INVALID_PARAMETER
A parameter passed was not correct.

6506 - MMIOERR_INTERNAL_SYSTEM
The operation failed due to an internal system error.

6507 - MMIOERR_NO_CORE
The system could not allocate the requested memory.

6508 - MMIOERR_INI_OPEN
The system could not open the MMPMMIO.INI file.

6509 - MMIOERR_INI_READ
The system could not read the MMPMMIO.INI file.

6510 - MMIOERR_INVALID_BUFFER_LENGTH
The specified buffer length is invalid.

6511 - MMIOERR_NO_BUFFER_ALLOCATED
Write operation expected a buffer but none was allocated.

6512 - MMIOERR_NO_FLUSH_FOR_MEM_FILE

A flush was requested for a MEM file.

6513 - MMIOERR_NO_FLUSH_NEEDED

A flush was requested, but the buffer was not dirty.

6514 - MMIOERR_READ_ONLY_FILE

The file was not opened in the WRITE mode.

6515 - MMIOERR_WRITE_ONLY_FILE

The file was not opened in the READ mode.

6516 - MMIOERR_INSTALL_PROC_FAILED

Installation of the I/O proc failed.

6517 - MMIOERR_READ_FAILED

System was unable to read, possible hardware error.

6518 - MMIOERR_WRITE_FAILED

Writing a dirty buffer before reading has failed.

6519 - MMIOERR_SEEK_FAILED

System was unable to seek, possible hardware error.

6520 - MMIOERR_CANNOTEXPAND

Unable to expand a MEM file for an advance request.

6521 - MMIOERR_FREE_FAILED

System was unable to free memory it allocated.

6522 - MMIOERR_EOF_SEEN

The system has reached the end of file.

6523 - MMIOERR_INVALID_ACCESS_FLAG

The specified access flag is invalid.

6524 - MMIOERR_INVALID_STRUCTURE

The specified structure is invalid.

6525 - MMIOERR_INVALID_SIZE

The specified size is invalid.

6526 - MMIOERR_INVALID_FILENAME

The specified filename is invalid.

6527 - MMIOERR_CF_DUPLICATE_SEEN

System located a duplicate CTOC entry.

6528 - MMIOERR_CF_ENTRY_NO_CORE

System could not allocate memory for CTOC entry.

6529 - MMIOERR_CF_WO_UNSUPPORTED

RIFF compound files cannot be opened write-only.

6530 - MMIOERR_CF_ELEMENTS_OPEN

Compound file cannot be closed because elements are open.

6531 - MMIOERR_CF_NON_BND_FILE

The specified file is not a RIFF compound file.

6532 - MMIOERR_CF_ENTRY_NOT_FOUND

System failed to find CTOC entry.

6533 - MMIOERR_DELETE_FAILED

System failed to delete the requested file.

6534 - MMIOERR_OUTOFMEMORY

The advance operation requires a buffer.

6535 - MMIOERR_INVALID_DLLNAME

The specified DLL name is incorrect.

6536 - MMIOERR_INVALID_PROCEDURENAME

The specified procedure is not valid for the DLL given.

6537 - MMIOERR_MATCH_NOT_FOUND
System could not locate a match in file entries.

6538 - MMIOERR_SEEK_BEFORE_BEGINNING
System cannot seek before beginning of file.

6539 - MMIOERR_INVALID_FILE
The specified filename is invalid.

6540 - MMIOERR_QOSUNAVAILABLE
Quality of Service is unavailable.

6541 - MMIOERR_MEDIA_NOT_FOUND
Media type not found on open.

6542 - MMIOERR_ERROR_IN_FRAME_DATA
The video frame data contains an error.

6543 - MMIOERR_INVALID_DIM_ALIGN
The video dimensions do not match the alignment.

6544 - MMIOERR_CODEC_NOT_SUPPORTED
The CODEC cannot be found.

6545 - MMIOERR_UNSUPPORTED_FUNCTION
The MMIO procedure does not support this function.

6546 - MMIOERR_CLIPBRD_ERROR
An unrecoverable error occurred while attempting to access the clipboard.

6547 - MMIOERR_CLIPBRD_ACTIVE
The file cannot be saved with its original name because there is an active reference to its data in the clipboard. Saving the file with its original name will cause this data to be lost.

6548 - MMIOERR_CLIPBRD_EMPTY
There is no compatible data in the clipboard for use by the paste operation.

6549 - MMIOERR_NEED_NEW_FILENAME
The file cannot be saved with its original name because there are other processes that have outstanding paste operations using the data in this file. Saving the file with its original name will cause this data to be lost.

6550 - MMIOERR_INCOMPATIBLE_TRACK_OPERATION
Reserved for future use.

6551 - MMIOERR_INCOMPATIBLE_DATA
The data in the clipboard cannot be pasted into this file because the characteristics of either the video or audio data, or both, do not match the characteristics of the target file.

6552 - MMIOERR_ACCESS_DENIED
The file system has denied access to the file for this operation.

6553 - MMIOERR_MISSING_FLAG
A required flag was not supplied with the command.

6554 - MMIOERR_INVALID_ITEM_FLAG
One or more of the item flags specified are invalid for this command.

Note: Base 7500 is reserved for user-defined error codes.

DIVE Return Codes

0x00000000 - DIVE_SUCCESS
Command completed successfully.

0x00001000 - DIVE_ERR_INVALID_INSTANCE
The DIVE instance handle specified in the *hDiveInst* parameter is invalid.

0x00001001 - DIVE_ERR_SOURCE_FORMAT
The FOURCC of the source format is not a recognized FOURCC.

0x00001002 - DIVE_ERR_DESTINATION_FORMAT
The FOURCC of the destination format is not a recognized FOURCC.

0x00001003 - DIVE_ERR_BLITTER_NOT_SETUP
[DiveSetupBlitter](#) must be called before a call is made to [DiveBlitImage](#).

0x00001004 - DIVE_ERR_INSUFFICIENT_LENGTH
The *pFormatData* of length *ulFormatLength* (specified in the [DIVE_CAPS](#) structure) is not large enough for the total number of input and output formats.

0x00001005 - DIVE_ERR_TOO_MANY_INSTANCES
There are not enough resources for another DIVE instance.

0x00001006 - DIVE_ERR_NO_DIRECT_ACCESS
The display adapter, display driver, or current video mode does not support direct-to-screen access.

0x00001007 - DIVE_ERR_NOT_BANK_SWITCHED
The display adapter is not a bank-switched adapter.

0x00001008 - DIVE_ERR_INVALID_BANK_NUMBER
The specified bank number is invalid.

0x00001009 - DIVE_ERR_FB_NOT_ACQUIRED
The frame buffer was not acquired in this instance.

0x0000100A - DIVE_ERR_FB_ALREADY_ACQUIRED
The frame buffer has already been acquired by this instance.

0x0000100B - DIVE_ERR_ACQUIRE_FAILED
The acquire action did not complete successfully.

0x0000100C - DIVE_ERR_BANK_SWITCH_FAILED
The bank could not be switched to the specified parameters.

0x0000100D - DIVE_ERR_DEACQUIRE_FAILED
The deacquire action did not complete successfully.

0x0000100E - DIVE_ERR_INVALID_PALETTE
The palette specified for the source or destination data is invalid.

0x0000100F - DIVE_ERR_INVALID_DESTINATION_RECTL
The rectangle of the output window is an invalid region.

0x00001010 - DIVE_ERR_INVALID_BUFFER_NUMBER
The buffer number must be a previously allocated or associated buffer number before it can be accessed.

0x00001011 - DIVE_ERR_SSMDD_NOT_INSTALLED
The device driver SSMDD.SYS is missing from CONFIG.SYS.

0x00001012 - DIVE_ERR_BUFFER_ALREADY_ACCESSED
The buffer has already been accessed by a previous call.

0x00001013 - DIVE_ERR_BUFFER_NOT_ACCESSED
The buffer has not been accessed (occurs on systems with accelerator-enabled hardware).

0x00001014 - DIVE_ERR_TOO_MANY_BUFFERS
There are not enough resources for another DIVE buffer.

0x00001015 - DIVE_ERR_ALLOCATION_ERROR
The hardware blitter memory allocation failed.

0x00001016 - DIVE_ERR_INVALID_LINESIZE
The scan line size specified in [DiveAllocImageBuffer](#) is invalid.

0x00001017 - DIVE_ERR_FATAL_EXCEPTION
DIVE is unable to register the exception handler to handle bank switching.

0x00001018 - DIVE_ERR_INVALID_CONVERSION

The source image format cannot be converted as requested to the destination image format.

0x00001019 - DIVE_ERR_VSD_ERROR

The vendor-specific driver (VSD) handle is invalid.

0x0000101A - DIVE_ERR_COLOR_SUPPORT

DIVE does not support any of the available hardware formats (occurs on overlay-accelerated systems).

0x0000101B - DIVE_ERR_OUT_OF_RANGE

Currently, DIVE_ERR_OUT_OF_RANGE is not returned by any function.

0x00001100 - DIVE_WARN_NO_SIZE

No blitting will occur because either the destination width or height is zero.

Notices

August 1996

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

This publication was produced in the United States of America. IBM may not offer the products, services, or features discussed in this document in other countries, and the information is subject to change without notice. Consult your local IBM representative for information on the products, services, and features available in your area.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate OS/2 programming techniques. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "(C) (your company name) (year). All rights reserved."

(C)Copyright International Business Machines Corporation 1994, 1996. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Disclaimers

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in

conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department LZKS, 11400 Burnet Road, Austin, TX 78758 U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

Common User Access	PAL
CUA	Presentation Manager
IBM	Ultimotion
Multimedia Presentation Manager/2	Ultimedia
OS/2	

The following terms are trademarks of other companies:

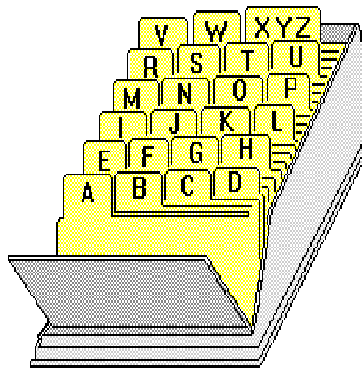
ActionMedia	Intel Corporation
Indeo	Intel Corporation
RealMagic	Sigma Designs, Inc.
Video Blaster	Creative Technology Ltd.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Glossary

Glossary

Select the starting letter of the glossary term you want to locate.



A

AB roll - Synchronized playback of two recorded video images so that one can perform effects, such as dissolves, wipes, or inserts, using both images simultaneously.

ACPA - Audio capture and playback adapter.

active matrix - A technology that gives every pel (dot) on the screen its own transistor to control it more accurately. (This allows for better contrast and less motion smearing.)

adaptive differential pulse code modulation - A bit-rate reduction technique where the difference in pulse code modulation samples are not compressed before being stored.

ADC - Analog-to-digital converter.

ADPCM - Adaptive differential pulse code modulation.

aliasing - The phenomenon of generating a false (alias) frequency, along with the correct one, as an artifact of sampling a signal at discrete points. In audio, this produces a "buzz." In imagery, this produces a jagged edge, or stair-step effect. See also *moire*.

all points addressable (APA) - In computer graphics, pertaining to the ability to address and display or not display each picture element on a display surface.

AM - Animation metafile.

ambience - In audio, the reverberation pattern of a particular concert hall, or listening space.

ambient noise - In acoustics, the noise associated with a particular environment, usually a composite of sounds from many distant or nearby sources.

American National Standards Institute (ANSI) - An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

AMF - Animation metafile format.

amp - See *amplifier*.

amplifier - (1) A device that increases the strength of input signals (either voltage or current). (2) Also referred to as an *amp*.

amp-mixer - (1) A combination amplifier and mixer that is used to control the characteristics of an audio signal from one or more audio sources. (2) Also referred to as an amplifier-mixer.

analog - Pertaining to data consisting of continuously variable physical quantities. Contrast with *digital*.

analog audio - Audio in which all information representing sounds is stored or transmitted in a continuous-scale electrical signal, such as line level audio in stereo components. See also *digital audio*.

analog-to-digital converter (ADC) - A functional unit that converts data from an analog representation to a digital representation. (I) (A)

analog video - Video in which all the information representing images is in a continuous-scale electrical signal for both amplitude and time.
See also *digital video*.

analog video overlay - See *overlay*.

anchor - An area of a display screen that is activated to accept user input. Synonymous with *hot spot*, *touch area*, and *trigger*.

animate - Make or design in such a way as to create apparently spontaneous, lifelike movement.

animated - Having the appearance of something alive.

animated screen capture - Recording a computing session for replay on a similar computer with voice annotation. (An example is sending a spreadsheet with an accompanying screen recording as an explanation and overview.)

animatic - A limited animation consisting of artwork shot on film or videotape and edited to serve as an on-screen storyboard.

animation metafile - A compound file format, the elements of which are the frames of animation themselves. These frames are stored sequentially so that they can be played back in time by streaming to the video agent.

animation metafile format (AMF) - The file format used to store animated frame sequences.

annotation - The linking of an object with another, where the second contains some information related to the first. For example, an audio annotation of a spreadsheet cell might contain verbal explanation about the contents of the cell.

ANSI - American National Standards Institute.

anthropomorphic software agent - The concept of a simulated agent, seemingly living inside the computer, that talks to and listens to the user, and then acts for the user on command.

antialiasing - (1) In imagery, using several intensities of colors (a ramp) between the color of the line and the background color to create the effect of smoother curves and fewer jagged edges on curves and diagonals. (2) In imagery or audio, removing aliases by eliminating frequencies above half the sample frequencies.

AOCA - Audio Object Content Architecture.

APA - All points addressable.

APA graphics - All Points Addressable graphics. See *bitmap graphics*.

API - Application programming interface.

application programming interface (API) - A functional interface supplied by the operating system or an IBM separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

application-supplied video window - (1) An application can specify to MMPM/2 that it wants video played in a specific window (controlled by the application) instead of the default window (controlled by MMPM/2). The application supplied video window can be used to implement advanced features not supported by the default video window. (2) Also referred to as an *alternate video window*.

artifact - A product resulting from human activity; in computer activity, a (usually unwanted) by-product of a process.

aspect ratio - (1) On a display screen, the ratio of the maximum length of a display line to the maximum length of a display column. (2) The ratio of height to width. (This term applies to areas or individual pels.) Refer to *enhanced graphics adapter* and *video graphics adapter*.

asymmetric video compression - In multimedia applications, the use of a powerful computer to compress a video for mastering so that a less powerful (less expensive) system is needed to decompress it. Contrast with *symmetric video compression*.

audible cue - A sound generated by the computer to draw a user's attention to, or provide feedback about, an event or state of the computer. Audible cues enhance and reinforce visible cues.

audio - Pertaining to the portion of recorded information that can be heard.

audio attribute control - Provides access to and operation of the standard audio attributes: mute, volume, balance, treble, and bass. All device communication and user interface support is handled by the control.

audio attributes - Refers to the standard audio attributes: mute, volume, balance, treble and bass.

audio clip - A section of recorded audio material.

Audio Object Content Architecture - A data format for multimedia products.

audio processing - Manipulating digital audio to, for example, edit or create special effects.

audio segment - A contiguous set of recorded data from an audio track. An audio segment might or might not be associated with a video segment.

audio track - (1) The audio (sound) portion of the program. (2) The physical location where the audio is placed beside the image. (A system with two audio tracks can have either stereo sound or two independent audio tracks.) (3) Synonym for *sound track*.

audiovisual - A generic term referring to experiences, equipment, and materials used for communication that make use of both hearing and sight.

Audio Visual Connection* (AVC) - An authoring system used on an IBM PS/2* to develop and display audiovisual productions.

audiovisual computer program - A computer program that makes use of both hearing and sight.

authoring - A structured approach to combining all media elements within an interactive production, assisted by computer software designed for this purpose.

authoring system - A set of tools used to create an interactive multimedia application without implementing formal programming.

AVI file format - The Audio/Video Interleaved (AVI) file format is the standard file format used to support software motion video. AVI files can contain multiple streams (tracks) of data (for example, a video and an audio stream). The streams are interleaved to improve access times during playback. The present implementation is limited to a single video stream and a single, optional, audio stream.

B

background image - The part of a display image, such as a form overlay, that is not changed during a particular sequence of transactions. Contrast with *foreground image*.

background music - In videotaping, music that accompanies dialog or action.

balance - For audio, refers to the relative strength of the left and right channels. A balance level of 0 is left channel only. A balance level of 100 is right channel only.

basic input/output system (BIOS) - In an IBM personal computer, microcode that controls basic hardware operations, such as interactions with diskette drives, hard disk drives, and the keyboard. (For example, the IBM Enhanced Graphics Adapter has an addressable BIOS, located on the adapter itself, that is used to control the IBM InfoWindow* graphics functions.)

beta format - A consumer and industrial 0.5-inch tape format.

BG - Script abbreviation for *background*.

BIOS - Basic input/output system.

bitmap - A coded representation in which each bit, or group of bits, represents or corresponds to an item, for example, a configuration of bits in main storage in which each bit indicates whether a peripheral device or a storage block is available or in which each group of bits corresponds to one pel of a display image.

bit-block transfer - Transfer of a rectangular array of bitmap data.

bitblt - Bit-block transfer. Synonym for *blit*.

bitmap graphics - A form of graphics whereby all points on the display are directly addressable. See also *all points addressable*.

blit - Synonym for *bitblt*.

blitter - Hardware that performs bit-block transfer operations.

BND - (1) An internal I/O procedure, provided by the MPPM/2 system, that supports RIFF compound files (commonly called *bundle files*). (2) The four-character code (*FOURCC*) of a bundle file. (3) See also *RIFF compound file*.

BND file - A RIFF compound file.

BND IOProc - An internal I/O procedure, provided by the MPPM/2 system, that supports the elements in a RIFF compound file. See also *CF IOProc*.

boom - A long, relatively horizontal supporting brace used for holding a microphone or camera; sometimes used to refer to the machinery

that supports the camera and allows it to move while shooting.

brightness - Refers to the level of luminosity of the video signal. A brightness level of 0 produces a maximally white signal. A brightness level of 100 produces a maximally black signal.

buffer - A portion of storage used to hold input or output data temporarily.

bundle file (BND) - (1) A file that contains many individual files, called *file elements*, bound together. The MMIO file manager provides services to locate, query, and access file elements in a bundle file. (2) A RIFF compound file.

bus - A facility for transferring data between several devices located between two end points, only one device being able to transmit at a given moment.

buy - (1) In videotaping, footage that is judged acceptable for use in the final video. (2) Synonym for *keeper*.

C

CAI - Computer-assisted instruction. Synonym for *CBT*.

calibration - The adjustment of a piece of equipment so that it meets normal operational standards. (For example, for the IBM InfoWindow system, calibration refers to touching a series of points on the screen so that the system can accurately determine their location for further reference.)

camcorder - A compact, hand-held video camera with integrated videotape recorder.

capture - To take a snapshot of motion video and retain it in memory. The video image may then be saved to a file or restored to the display.

cast animation - (1) A sequence of frames consisting of manipulations of graphical objects. (2) The action of bringing a computer program, routine, or subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (3) See also *frame animation*.

CAV - Constant angular velocity.

CBT - Computer-based training. Synonym for *CAI*.

CCITT - Comite Consultatif International Telegraphique et Telephonique. The International Telegraph and Telephone Consultative Committee.

CD - Compact disc.

CD-DA - Compact disc, digital audio.

CD-I - Compact Disc-Interactive.

CD-ROM - Compact disc, read-only memory.

CD-ROM XA - Compact disc, read-only memory extended architecture.

cel - A single frame (display screen) of an animation. (The term originated in cartooning days when the artist drew each image on a sheet of celluloid film.)

CF IOProc - An internal I/O procedure, provided by the MMPM/2 system, that supports RIFF compound files. The CF IOProc operates on the entire compound file (rather than on the elements in a RIFF compound file, as with the BND IOProc). The CF IOProc is limited to operations required by the system to ensure storage system transparency at the application level. See also *BND IOProc*.

CGA - Color graphics adapter.

CGRP - Compound file resource group.

channel mapping - The translation of a MIDI channel number for a sending device to an appropriate channel for a receiving device.

channel message - A type of non-SysEx MIDI message that has a channel identifier in it, implying that these messages are specific to one channel.

check disc - A videodisc produced from the glass master that is used to check the quality of the finished *interactive program*.

chord - To press more than one button at a time on a pointing device.

chroma-key color - The specified first color in a combined signal. See also *chroma-keying*.

chroma-keying - Combining two video signals that are in sync. The combined signal is the second signal whenever the first is of some specified color, called the chroma-key color, and is the first signal otherwise. (For example, the weatherman stands in front of a blue background-blue is the chroma-key color.) At home, the TV viewer sees the weather map in place of the chroma-key color, with the weatherman suspended in front.

chroma signal - The portion of image information that provides the color (hue and saturation).

chrominance - The difference between a color and a reference white of the same luminous intensity.

chunk - (1) The basic building block of a RIFF file. (2) A RIFF term for a formalized data area. There are different types of chunks, depending on the chunk ID. (3) See *LST chunk* and *RIFF chunk*.

chunk ID - A four-character code (FOURCC) that identifies the representation of the chunk data.

circular slider control - A knob-like control that performs like a control on a TV or stereo.

circular slider knob - A knob-like dial that operates like a control on a television or stereo.

class - (1) A categorization or grouping of objects that share similar behaviors and characteristics. Synonym for *object class*. (2) See *node class*. (RTMIDI-specific term)

click - To press and release a button on a pointing device without moving the pointer off the choice. See *double-click*. See also *drag select*.

clip - A section of recorded, filmed, or videotaped material. See also *audio clip* and *video clip*.

closed circuit - A system of transmitting television signals from a point of origin to one or many restricted destination points specially equipped to receive the signals.

close-up - In videotaping, the picture obtained when the camera is positioned to show only the head and shoulders of a subject; in the case of an object, the camera is close enough to show details clearly. See also *extreme close-up*.

CLP - Common loader primitive.

CLUT - Color look-up table. Synonym for *color palette*.

CLV - Constant linear velocity.

CODEC - compressor/decompressor (CODEC) - An algorithm implemented either in hardware or software that can either compress or decompress a data object. For example, a CODEC can compress raw digital images into a smaller form so that they use less storage space. When used in the context of playing motion video, decompressors reconstruct the original image from the compressed data. This is done at a high rate of speed to simulate motion.

collaborative document production - A system feature that provides the ability for a group of people to manage document production.

collision - An unwanted condition that results from concurrent transmissions on a channel. (T) (For example, an overlapping condition that occurs when one sprite hides another as it passes over it.)

color cycling - An animation effect in which colors in a series are displayed in rapid succession.

color graphics adapter (CGA) - An adapter that simultaneously provides four colors and is supported on all IBM Personal Computer and Personal System/2* models.

colorization - The color tinting of a monochrome original.

color palette - (1) A set of colors that can be displayed on the screen at one time. This can be a standard set used for all images or a set that can be customized for each image. (2) Synonym for *CLUT*. (3) See also *standard palette* and *custom palette*.

common loader primitive - A system service that provides a high-level interface to hardware-specific loaders.

common user access (CUA) - (1) Guidelines for the dialog between a human and a workstation or terminal. (2) One of the three SAA architectural areas.

compact disc (CD) - A disc, usually 4.75 inches in diameter, from which data is read optically by means of a laser.

compact disc, digital audio (CD-DA) - The specification for audio compact discs. See also *Redbook audio*.

Compact Disc-Interactive (CD-I) - A low-cost computer, being developed by N.V. Phillips (The Netherlands) and Sony (Japan), that plugs into standard television sets to display text and video stored on compact discs.

compact disc, read-only memory (CD-ROM) - High-capacity, read-only memory in the form of an optically read compact disc.

compact disc, read-only memory extended architecture (CD-ROM XA) - An extension to CD-ROM supporting additional audio and video levels for compression and interlacing of audio, video, and digital data.

component video - A video signal using three signals, one of which is luminance, and the other two of which are the color vectors. See also *composite video* and *S-video*.

composed view - A view of an object in which relationships of the parts contribute to the overall meaning. Composed views are provided primarily for data objects.

composite - The combination of two or more film, video, or electronic images into a single frame or display. See also *composite video*.

composite monitor - A monitor that can decode a color image from a single signal, such as NTSC or PAL. Contrast with *RGB*.

composite object - An object that contains other objects. For example, a document object that contains not only text, but graphics, audio, image, and/or video objects, each of which can be manipulated separately as an individual object.

composite video - A single signal composed of chroma, luminance, and sync. NTSC is the composite video that is currently the U.S. standard for television. See also *component video* and *S-video*.

compound device - A multimedia device model for hardware that requires additional data objects, referred to as data elements, before multimedia operations can be performed.

compound file - A file that contains multiple file elements.

compound file resource group (CGRP) - A RIFF chunk that contains all the compound file elements, concatenated together.

compound file table of contents (CTOC) - A *RIFF chunk* that indexes the CGRP chunk, which contains the actual multimedia data elements. Each entry contains the name of, and other information about, the element, including the offset of the element within the CGRP chunk. All the *CTOC* entries of a table are of the same length and can be specified when the file is created.

compound message - A structure which combines a time stamp, a source instance identifier, a track number, and a MIDI message. Each of these fields is 32 bits, so the structure is 16 bytes in length. (RTMIDI-specific term)

computer-animated graphics - Graphics animated by using a computer, compared to using videotape or film.

computer-assisted instruction (CAI) - (1) A data processing application in which a computing system is used to assist in the instruction of students. The application usually involves a dialog between the student and a computer program. An example is the OS/2 tutorial. (2) Synonym for *computer-based training*.

computer-based training (CBT) - Synonym for *computer-assisted instruction*.

computer-controlled device - An external video source device with frame-stepping capability, usually a videodisc player, whose output can be controlled by the multimedia subsystem.

conforming - Performing final editing on film or video using an offline edited master as a guide.

connection - The establishment of the flow of information from a connector on one device to a compatible connector on another device. A connection can be made that is dependent on a physical connection, for example the attachment of a speaker to an audio adapter with a speaker wire. A connection can also be made that is completely internal to the PC, such as the connection between the waveaudio media device and the ampmix device. See also *connector*.

connector - A software representation of the physical way in which multimedia data moves from one device to another. A connector can have an external representation, such as a headphone jack on a CD-ROM player. A connector can also have an internal representation, such as the flow of digital information into an audio adapter. See also *connection*.

constant angular velocity (CAV) - Refers to both the format of data stored on a videodisc and the videodisc player rotational characteristics. CAV videodiscs contain 1 frame per track. This allows approximately 30 minutes of playing time per videodisc side. CAV videodisc players spin at a constant rotational speed (1800 rpm for NTSC or 1500 rpm for PAL) and play 1 frame per disc revolution. CAV players support *frame-accurate searches*. See also *constant linear velocity*.

constant linear velocity (CLV) - Refers to both the format of data stored on a videodisc and the videodisc player characteristics. CLV videodiscs contain 1 frame on the innermost track and 3 frames of data on the outermost track. This allows approximately 1 hour of playing time per videodisc side. CLV videodisc players vary the rotational speed from approximately 1800 rpm at the inner tracks to 600 rpm at the outer tracks (for NTSC).

Currently, few CLV players support *frame-accurate searches*. They only support search or play to within one second (30 frames for NTSC or 25 frames for PAL). See also *constant angular velocity*.

container - An object whose specific purpose is to hold other objects. A folder is an example of a container object.

contents view - A view of an object that shows the contents of the object in list form. Contents views are provided for container objects and for any object that has container behavior, for example, a device object such as a printer.

continuity - In videotaping, consistency maintained from shot to shot and throughout the take. For example, a switch that is on in one shot should not be off in the next unless it was shown being turned off.

continuous media object - A data object that varies over time; a stream-oriented data object. Examples include audio, animation, and video.

contrast - The difference in brightness or color between a display image and the area in which it is displayed. A contrast level of 0 is minimum difference. A contrast level of 100 is maximum difference.

control - A visual user interface component that allows a user to interact with data.

coordinate graphics - (1) Computer graphics in which display images are generated from display commands and coordinate data. (2) Contrast with *raster graphics*. (3) Synonym for *line graphics*.

crop - To cut off; to trim (for example, a tape).

crossfade - Synonym for *dissolve*.

cross-platform - Used to describe applications that are operable with more than one operating system.

cross-platform transmission - Electronic transmission of information (such as mail) between incompatible operating systems.

CTOC - Compound file table of contents.

CU - Script abbreviation for *close-up*.

CUA - Common User Access.

cue point - A point that the system recognizes as a signal that may be acted upon.

custom palette - (1) A set of colors that is unique to one image or one application. (2) See also *standard palette* and *color palette*.

cut - The procedure of instantly replacing a picture from one source with a picture from another. (This is the most common form of editing scene to scene.)

D

DAC - Digital-to-analog converter.

data object - In an application, an element of a data structure (such as a file, an array, or an operand) that is needed for program execution and that is named or otherwise specified by the allowable character set of the language in which the program is coded.

data stream - All data transmitted through a data channel.

data streaming - Real-time, continuous flowing of data.

DCP - See *device control panel*.

decode - (1) To convert data by reversing the effect of previous encoding. (2) To interpret a code. (3) To convert encoded text into plaintext by means of a code system. (4) Contrast with *encode*.

default video window - (1) Refers to where video is displayed when an application does not indicate an application-defined window with the MCI_WINDOW message. This is provided by and managed for the application by MMPM/2. (2) See also *application-defined window*.

default window - See *default video window*.

delta frame - Refers to one or more frames occurring between reference frames in the output stream. Unlike a reference frame, which stores a complete image, a delta frame stores only the changes in the image from one frame to the next. See *reference frame*.

destination rectangle - An abstract region which defines the size of an image to be created when recording images for software motion video playback. The ratio of this rectangle's size to that of the source rectangle determines the scaling factor to be applied to the video.

destination window - See *destination rectangle*.

device capabilities - The functionality of a device, including supported component functions.

device context - The device status and characteristics associated with an opened instance of an Media Control Interface device.

device control panel (DCP) - An integrated set of controls that is used to control a device or media object (such as by playing, rewinding, increasing volume, and so on).

device controls - See *standard multimedia device controls*.

device driver - (1) A file that contains the code needed to use an attached device. (2) A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD drive.

device element - A data object, such as a file, utilized by a compound device.

device object - An object that provides a means for communication between a computer and the outside world. A printer is an example of a device object.

device sharing - (1) The ability to share a device among many different applications simultaneously. If a device is opened shareable, the device context will be saved by the operating system when going from one application to another application. (2) Allowing a device context to be switched between Media Control Interface devices.

device-specific format - The storage or transmission format used by a device, especially if it is different from an accepted standard.

dialog - In an interactive system, a series of related inquiries and responses similar to a conversation between two people.

digital - (1) Pertaining to data in the form of numeric characters. (2) Contrast with *analog*.

digital audio - (1) Material that can be heard that has been converted to digital form. (2) Synonym for *digitized audio*.

digital signal processor (DSP) - A high-speed coprocessor designed to do real-time manipulation of signals.

digital video - (1) Material that can be seen that has been converted to digital form. (2) Synonym for *digitized video*.

digital video effects (DVE) - An online editing technique that manipulates on-screen a full video image; activity for creating sophisticated transitions and special effects. Digital video effects (DVE) can involve moving, enlarging, or overlaying pictures.

Digital Video Interactive (DVI) - A system for bringing full-screen, full-motion television pictures and sound to a regular PC. DVI is a chip set and uses delta compression; that is, only the image-to-image changes in each frame are saved rather than the whole frame. Data (video footage) is compressed into a form that reduces memory requirements by factors of 100 or greater. This compressed data is stored on optical discs and can be retrieved at a rate of 30 frames per second. (The DVI technology was developed by RCA and then sold to Intel. IBM has chosen this technology for future use in the PS/2.)

digital-to-analog converter (DAC) - (1) A functional unit that converts data from a digital representation to an analog representation. (2) A device that converts a digital value to a proportional analog signal.

digitize - To convert an analog signal into digital format. (An analog signal during conversion must be *sampled* at discrete points and quantized to discrete numbers.)

digitized audio - Synonym for *digital audio*.

digitized video - Synonym for *digital video*.

digitizer - A device that converts to digital format any image captured by the camera.

direct manipulation - A set of techniques that allow a user to work with an object by dragging it with a pointing device or interacting with its pop-up menu.

direct memory access - The transfer of data between memory and input and output units without processor intervention.

direct-read-after-write (DRAW) disc - A videodisc produced directly from a videotape, one copy at a time. A DRAW disc usually is used to check program material and author applications before replicated discs are available.

disc - Alternate spelling for *disk*.

discard stop - In data streaming, requests that the data stream be stopped and the data remaining in the stream buffers be discarded.

disk - A round, flat, data medium that is rotated in order to read or write data.

display image - A collection of display elements or segments that are represented together at any one time on a display surface. See also *background image* and *foreground image*.

dissolve - To fade down one picture as the next fades up. Synonym for *crossfade*.

dithering - When different pixels in an image are prebiased with a varying threshold to produce a more continuous gray scale despite a limited palette. This technique is used to soften a color line or shape. This technique also is used for alternating pixel colors to create the illusion of a third color.

DLL - Dynamic-link library.

dolly - A wheeled platform for a camera; a camera movement where the tripod on which the camera is mounted physically moves toward or away from the subject.

DOS IOProc - An internal I/O procedure, provided by the MMPM/2 system, that supports DOS files.

double-click - In SAA Advanced Common User Access, to press and release a mouse button twice within a time frame defined by the user, without moving the pointer off the choice. See *click*. See also *drag select*.

drag select - In SAA Advanced Common User Access, to press a mouse button and hold it down while moving the pointer so that the pointer travels to a different location on the screen. Dragging ends when the mouse button is released. All items between the button-down and button-up points are selected. See also *click*, *double-click*.

DRAW disc - Direct-read-after-write disc.

drop-frame time code - A nonsequential time code used to keep tape time code matched to real time. Must not be used in tapes intended for videodisc mastering.

DSP - Digital signal processor.

DTMF - Dual-tone modulation frequency.

dual plane video system - Refers to when graphics from the graphics adapter are separate from the analog video. That is, there is a separate graphics plane and video plane. The analog video appears behind the graphics, showing through only in the areas that are transparent. Since graphics and video are separate, capturing the graphics screen will only obtain graphics, and capturing the video screen will only obtain video. This is also true for restoring images. See also *single plane video system*.

dual-state push button - A push button that has two states, in and out. It is used for setting and resetting complementary states, such as Mute and Unmute.

dual-tone modulation frequency (DTMF) - Pushbutton phone tones.

dub - To copy a tape; to add (sound effects or new dialog) to a film; to provide a new audio track of dialog in a different language. (Often used with "in" as "dub in".)

DVE - Digital video effects.

DVI - Digital Video Interface

dynamic icon - An icon that changes to convey some information about the object that it represents. For example, a folder icon can show a count, indicating the number of objects contained within the folder. Also, a tape player icon can show an animation of turning wheels to indicate that the machine playing.

dynamic linking - In the OS/2 operating system, the delayed connection of a program to a routine until load time or run time.

dynamic link library (DLL) - A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic resource allocation - An allocation technique in which the resources assigned for execution of computer programs are determined by criteria applied at the moment of need. (I) (A)

dynamic resource - A multimedia program unit of data that resides in system memory. Contrast with *static resource*.

E

earcon - An icon with an audio enhancement, such as a ringing telephone.

ECB - Event control block.

ECU - Script abbreviation for *extreme close-up*.

edit decision list (EDL) - Synonym for *edit list*.

edit list - A list of the specific video footage, with time-code numbers, that will be edited together to form the program. It is completed during the offline edit and used during the online edit. Synonym for *edit decision list (EDL)*.

edit master - The final videotape from which all copies are made. See also *glass master*.

editing - Assembling various segments into the composite program.

EDL - Edit decision list.

EGA - Enhanced graphics adapter.

element - (1) A file or other stored data item. (2) An individual file that is part of a RIFF compound file. An element of a compound file also could be an entire RIFF file, a non-RIFF file, an arbitrary RIFF chunk, or arbitrary binary data. (3) The particular resource within a subarea that is identified by an element address.

emphasis - Highlighting, color change, or other visible indication of the condition of an object or choice and the effect of that condition on a user's ability to interact with that object or choice. Emphasis can also give a user additional information about the state of an object or choice.

encode - To convert data by the use of a code in such a manner that reconversion to the original form is possible. Contrast with *decode*.(T)

enhanced graphics adapter (EGA) - A graphics controller for color displays. The pel resolution of an enhanced graphics adapter is 3:4.

entry field - An area into which a user places text. Its boundaries are usually indicated.

erasable optical discs - Optical discs that can be erased and written to repeatedly.

establishing shot - In videotaping, a long shot used in the beginning of a program or segment to establish where the action is taking place and to give the sense of an environment.

EVCB - Event control block.

event - An occurrence of significance to a task; for example, the completion of an asynchronous operation, such as I/O.

event control block (ECB or EVCB) - A control block used to represent the status of an event.

event queue - In computer graphics, a queue that records changes in input devices such as buttons, valuator, and the keyboard. The event queue provides a time-ordered list of input events.

event semaphore - (1) Used when one or more threads must wait for a single event to occur. (2) A blocking flag used to signal when an event has occurred.

explicit event - An event supported by only some handlers, such as a custom event unique to a particular type of data.

EXT - Script abbreviation for *exterior*.

extended selection - A type of selection optimized for the selection of a single object. A user can extend selection to more than one object, if required. The two kinds of extended selection are contiguous extended selection and discontinuous extended selection.

extreme close-up - The shot obtained when the camera is positioned to show only the face or a single feature of the subject; in the case of an object, the camera is close enough to reveal an individual part of the object clearly.

F

facsimile machine - A functional unit that converts images to signals for transmission over a telephone system or that converts received signals back to images.

fade - To change the strength or loudness of a video or audio signal, as in "fade up" or "fade down."

fast threads - Threads created by an application that provide minimal process context, for example, just stack, register, and memory. With the reduced function, fast threads can be processed quickly.

FAX machine - Synonym for facsimile machine.

file cleanup - The removal of superfluous or obsolete data from a file.

file compaction - Any method of encoding data to reduce its required storage space.

file element - An individual file that is part of a RIFF compound file. An element of a compound file can also be an entire RIFF file, a non-RIFF file, an arbitrary RIFF chunk, or arbitrary binary data. See *media element*.

file format - A language construct that specifies the representation, in character form, of data objects in a file. For example, MIDI, M-Motion, or AVC.

file format handler - (1) I/O procedure. (2) Provides functions that operate on the media object of a particular data format. These functions include opening, reading, writing, seeking, and closing elements of a storage system.

file format IOProc - (1) An *installable I/O procedure* that is responsible for all technical knowledge of the format of a specific type of data, such as headers and data compression schemes. A file format IOProc manipulates multimedia data at the element level. A file format IOProc handles the element type it was written for and does not rely on any other file format IOProcs to do any processing. However, a file format IOProc might need to call a *storage system IOProc* to obtain data within a file containing multiple file elements. (2) See *IOProc*. (3) See also *storage system IOProc*.

filter - A certain type of node that modifies messages and forwards them. Filters are used to perform real-time processing of MIDI data. When a filter receives a message, it may perform some manipulation on it. It will then forward the message. (RTMIDI-specific term)

final script - The finished script that will be used as a basis for shooting the video. Synonym for *shooting script*.

first draft - A rough draft of the complete script.

first generation - In videotaping, the original or master tape; not a copy.

flashback - Interruption of chronological sequence by interjection of events occurring earlier.

flush stop - In data streaming, requests that the source stream handler be stopped but the target stream handler continue until the last buffer held at the time the stop was requested is consumed by the target stream handler.

flutter - A phenomenon that occurs in a videodisc freeze-frame when both video fields are not identically matched, thus creating two different pictures alternating every 1/60th of a second.

fly-by - Animation simulating a bird's-eye view of a three-dimensional environment.

fly-in - A DVE where one picture "flies" into another.

folder - A file used to store and organize documents or electronic mail.

footage - The total number of running feet of film used (as for a scene).

foreground image - The part of a display image that can be changed for every transaction. Contrast with *background image*.

form overlay - A pattern such as a report form, grid, or map used as background for a display image.

form type - A field in the first four bytes of the data field of a RIFF chunk. It is a four-character code identifying the format of the data stored in the file. A RIFF form is a chunk with a chunk ID of RIFF. For example, waveform audio files (WAVE files) have a form type of WAVE.

Format 0 MIDI file - All MIDI data is stored on a single track.

Format 1 MIDI file - All MIDI data is stored on multiple tracks.

forward - To re-transmit a message that was received. Each instance can have any number of links from it. When an instance receives a message, it may decide to send the same message along its links. This is known as forwarding. (RTMIDI-specific term)

four-character code (FOURCC) - A 32-bit quantity representing a sequence of one to four ASCII alphanumeric characters (padded on the right with blank characters). Four-character codes are unique identifiers that represent the file format and I/O procedure.

FOURCC - Four-character code.

fps - Frames per second.

frame - In film, a complete television picture that is composed of two scanned fields, one of the even lines and one of the odd lines. In the NTSC system, a frame has 525 horizontal lines and is scanned in 1/30th of a second.

frame-step recording - Refers to the capturing of video and audio data frame by frame, from a computer-controlled, frame-steppable video source device, or a previously recorded AVI file.

frame-accurate searches - The ability of a videodisc player to play or search to specific frames on the videodisc via software or remote control. This capability is available on all CAV players, but currently only on a few CLV players. Most CLV players can only search or play to within one second (30 frames for NTSC or 25 frames for PAL).

frame animation - A process where still images are shown at a constant rate. See also *cast animation*.

frame grabber - A device that digitizes video images.

frame number - The number used to identify a frame. On videodisc, frames are numbered sequentially from 1 to 54,000 on each side and can be accessed individually; on videotape, the numbers are assigned by way of the SMPTE time code.

frame rate - The speed at which the frames are scanned-30 frames a second in NTSC video, 25 frames a second in PAL video, and 24 frames a second in *mos* film.

A complete television picture frame is composed of two scanned fields, one of the even lines and one of the odd lines. In the NTSC system, a frame has 525 horizontal lines and is scanned in 1/30th of a second. In the PAL system, a frame has 625 horizontal lines and is scanned in 1/25th of a second.

freeze - Disables updates to all or part of the video buffer. The last video displayed remains visible. See also *unfreeze*.

freeze-frame - A frame of a motion-picture film that is repeated so as to give the illusion of a still picture.

full-frame time code - (1) A standardized method, set by the Society of Motion Picture and Television Engineers (SMPTE), of address coding a videotape. It gives an accurate frame count rather than an accurate clock time. (2) Synonym for *nondrop time code*.

full-motion video - Video playback at 30 frames per second for NTSC signals or 25 frames per second for PAL signals.

G

game port - On a personal computer, a port used to connect devices such as joysticks and paddles.

GDT - Global Descriptor Table.

general purpose interface bus - An adapter that controls the interface between the PC, Personal Computer XT*, or Personal Computer AT* and, for example, the InfoWindow display; also known as the IBM IEEE 488.

genlock - A device that comes on an adapter or plugs into a computer port and provides the technology to overlay computer-generated titles and graphics onto video images. It does this by phase-*locking* the sync *generation* of two video signals together so they can be merged. Genlock also converts a digital signal to NTSC or PAL format. (For example, flying logos and scrolling text on television shows are overlaid using a genlock.)

glass master - The final videodisc format from which copies are made.

Global Descriptor Table (GDT) - Defines code and data segments available to all tasks in an application.

GOCA - Graphic Object Content Architecture.

Graphic Object Content Architecture (GOCA) - (1) A data format for multimedia products. (2) A push button with graphic, two-state, and animation capabilities.

graphics overlay - The nature of dual plane video systems makes it possible to place graphics over video. Only the pels of the designated transparent color allows the video to show through. All other graphics pels appear on top of the video. Note that the video still exists in the video buffer under the non-transparent graphics pels.

graphics plane - In a dual plane video system, the graphics plane contains material drawn or generated by the graphics adapter. The graphics plane will be combined with the video plane to create an entire display image.

grayscale - See *greyscale*.

greyscale - When video is displayed in shades of black and white.

grouping - For Media Control Interface devices, refers to the ability to associate dissimilar devices for a common purpose. Grouping MCI

devices aids resource management by insuring that all devices in a group are kept together.

H

handshaking - The exchange of predetermined signals when a connection is established between two data set devices.

help view - A view of an object that provides information to assist users in working with that object.

Hi8 - High-band 8mm videotape format.

HID - Handler identification.

High Sierra Group (HSG) - (1) A group that set the standards for information exchange for a CD-ROM. (2) HSG also refers to those standards (HSG standards).

HMS - (1) Hours-minutes-seconds. (2) A time format for videodisc players.

HMSF - (1) Hours-minutes-seconds-frames. (2) A time format for videodisc players.

hot spot - The area of a display screen that is activated to accept user input. Synonym for *touch area*.

HSG - High Sierra Group.

HSI - Hue saturation intensity.

hue - Describes the position of a color in a range from blue to green. A hue level of 0 is maximum blue. A hue level of 100 is maximum green. Synonym for *tint*.

hue saturation intensity (HSI) - A method of describing color in three-dimensional color space.

HWID - Hardware identifier.

hypermedia - Navigation or data transfer between connected objects of different media types. For example, a user might navigate from an image object to an audio object that describes the image over a hypermedia link. Or, a representation of a graph object might be embedded in a document object with a hypermedia data transfer link, such that when the graph object is changed, the representation of that object in the document is also changed.

Hytime - An ANSI standard proposal that addresses the synchronization and linking of multimedia objects.

I

IDC - Inter-device communication mechanism provided by the OS/2 function ATTACHDD DevHelp.

identifier - (1) A sequence of bits or characters that identifies a program, device, or system to another program, device, or system. (2) In the C language, a sequence of letters, digits, and underscores used to identify a data object or function. (3) See *four-character code (FOURCC)*.

IDOCA - Integrated Data Object Content Architecture.

image - An electronic representation of a video still.

image bitsperpel - Pertaining to the number of colors supported by the current pel format. The currently accepted standard values are those supported by OS/2 bitmaps, for example, 1, 4, 8, or 24 bits per pel. In addition, 12 bits per pel formats are accepted for YUV images (including the 'yuvb' pel format for RDIB files).

image buffer - A location in memory where video images are stored for later use.

image buffer formats - The format or representation of data buffers containing video images.

image compression - The method of compressing video image data to conserve storage space.

image file format - The format or representation of data files containing video images.

Image Object Content Architecture (IOCA) - A data format for multimedia products.

image pelformat - Indicates the color representation that is to be used for images that are captured and saved. This normally includes palettized RGB, true-color RGB, or YUV color formats.

image quality - Represents the user's or application's subjective evaluation of complexity and quality of the image to be captured or saved. This setting is used to determine specific compression methods to use for saving the image.

implicit event - An event that all stream handlers always must support, such as *end of stream* or *preroll complete*.

in-betweening - Synonym for *tweening*.

in frame - Refers to a subject that is included within the frame. See also *frame*.

InfoWindow system - A display system that can combine text, graphics, and video images on a single display. The minimum system configuration is the IBM InfoWindow Color Display, a system unit, a keyboard, and one or two videodisc players.

input locking mask - A filter, or mask, that controls which areas of the display can display or freeze video.

input/output control (IOctl) - A system function that provides a method for an application to send device-specific control commands to a device driver.

installable I/O procedure - A file format handler that provides functions that operate on the media object of a particular data format. These functions include opening, reading, writing, seeking, and closing elements.

instance - See *node instance*. (RTMIDI-specific term)

INT - Script abbreviation for *interior*.

Integrated Data Object Control Architecture (IDOCA) - A data format for multimedia products.

interactive multimedia - The delivery of information content through combinations of video, computer graphics, sound, and text in a manner that allows the user to interact.

interactive program - A running program that can receive input from the keyboard or another input device. Contrast with *noninteractive program*.

interactive videodisc system (IVS) - A system in which a user can interact with a videodisc display image by entering commands to the computer through a device such as a keyboard or keypad or by touching a touch-sensitive screen at specific points on the display surface.

interlace flicker - The apparent flicker when one field of an interlaced image contains more light than the other field due to the placement of image details with respect to the separate fields. Two methods used to avoid interlace flicker: limit the vertical resolution on natural images (as opposed to text or graphics); design characters so that each character has an equal number of pels in each field.

interlacing - In multimedia applications, a characteristic of video image display that results in greater image clarity. In effect, the video image is traced across the screen twice. (The time delay between the two tracings makes this effect undesirable for normal computer-generated graphics.) Synonymous with *interleaving*.

interleaving - (1) The simultaneous accessing of two or more bytes or streams of data from distinct storage units. (2) The alternating of two or more operations or functions through the overlapped use of a computer facility. (3) In a duplicator, the process of inserting absorbent sheets between successive sheets of the copy paper to prevent set-off. (T) (4) Synonym for interlacing.

internal I/O procedure - An I/O procedure that is built in to the MMPM/2 system, including DOS, MEM, BND, and CF IOProcs.

International Organization for Standardization (ISO) - An organization of national standards bodies from various countries established to promote development of standards to facilitate international exchange of goods and services, and develop cooperation in intellectual, scientific, technological, and economic activity.

IOctl - Input/output control.

IOProc - A file format handler that provides functions that operate on the media object of a particular data format. These processes include opening, reading, writing, seeking, and closing elements of a storage system. There are two classes of I/O procedures: *file format* and *storage system*.

iris - To fade a picture by operating the iris (aperture) on the camera in a certain way; the type of computer image dissolve accomplished by operating the aperture in a certain way.

ISO - International Organization for Standardization.

ISP - IBM Signal Processor. An IBM proprietary digital signal processor.

ISPOS - IBM Signal Processor Operating System.

ISV - Independent software vendor.

items - Options, choices or keywords such as one or more of the following options, choices or keywords can or should be specified. Sometimes use of these items can also be exclusive or some items may not be compatible with other items.

IVS - Interactive videodisc system.

J

JIT - Just in time. (Often used with "learning" as "JIT learning".) Multimedia help functions, closely integrated with applications, that employ voice output to guide the user.

Joint Photographic Experts Group (JPEG) - A group that is working to establish a standard for compressing and storing still images in digital form. JPEG also refers to the standard under development by this group (JPEG standard).

joy stick - In computer graphics, a lever that can pivot in all directions and that is used as a locator device. (Resembles an airplane's joy stick).

JPEG - Joint Photographic Experts Group.

K

keeper - Synonym for *buy*.

kernel - (1) The part of an operating system that performs basic functions such as allocating hardware resources. (2) A program that can run under different operating system environments. (3) A part of a program that must be in main storage in order to load other parts of the program.

key frames - The start and end frames of a single movement in an animation sequence; also can refer to the periodic full-frame image interspersed in the stream to allow random starts from these full-frame images (key frames).

keypad - A small, often hand-held, keyboard.

L

laser - Light amplification by stimulated emission of radiation; the device that produces this light.

latency - In video, the time it takes for the light from the phosphor screen to decay after the excitation is removed. Long-persistence phosphor has less flicker of still images, but more blurring of moving images.

level one videodisc applications - Interactive applications based on manual keypad functions, picture stops, and chapter stops.

level three videodisc applications - Interactive applications controlled by an external computer that uses the videodisc player as a peripheral device.

level two videodisc applications - Interactive applications controlled by the keypad and the videodisc player's internal computer. The control program is recorded on the videodisc itself.

LIB - Dynamic-link definition library. A file containing the data needed to build a program .EXE file but which does not contain the dynamic-link programs themselves. Contrast with *dynamic-link library*.

light pen - A light-sensitive pick device that is used by pointing it at the display surface.

line graphics - Synonym for *coordinate graphics*.

line pairing - A faulty interlace pattern in which the lines of the second field begin to pair with the lines of the first field rather than fit exactly within them.

linear audio - The analog audio on the linear track of videotape that can be recorded without erasing existing video; used for audio dubbing after video is edited.

linear video - A sequence of motion footage played from start to finish without stops or branching, like a movie.

link - A one-way link (directed edge) from one instance to another. If an instance wishes to send a message, it is sent along all the links from it. An instance can have any number of links coming from it, and any number of other instances can have links to it. An instance cannot select along which links the message should be sent. See also *slot*. (RTMIDI-specific term)

LIST chunk - A chunk that contains a list or an ordered sequence of subchunks.

list type - (1) A field in the first four bytes of the data field of a LIST chunk. (2) A four-character code identifying the contents of the list.

locked memory - An area of memory that is not available for use because it is being held by another process.

long shot - (1) A camera angle that reveals the subject and the surroundings. Often used as an establishing shot. (2) Synonym for *wide shot*.

LS - Script abbreviation for *long shot*.

luminance signal - The portion of image information that provides brightness. Alone, luminance provides a monochrome image.

M

M-ACPA - M-Audio Capture and Playback Adapter.

M-Audio Capture and Playback Adapter (M-ACPA) - An adapter card (for use with the IBM PS/2 product line) that provides the ability to record and play back high quality sound. The adapter converts the audio input (analog) signals to a digital format that is compressed and stored for later use.

master stream handler - Controls the behavior of one or more subordinate objects (the *slave streams*).

matte - In film, an opaque piece of art or a model that leaves a selected area unexposed to be filled on a subsequent pass or in composite.

MCD - Media control driver.

MCI - Media Control Interface.

M-Control Program/2 - The software interface required for the M-Motion Video Adapter/A. It consists of APIs or toolkits for DOS, Windows, Windows MCI, OS/2, and OS/2 MMPM/2. It also includes Pioneer and Sony videodisc player drivers for these environments. See also *M-Motion Video Adapter/A*.

MDM - Media device manager.

media - More than one hardware medium.

media component - A processor of audiovisual information or media. Media components can be either internal or external physical devices or defined mechanisms for effecting higher-level function from internal hardware and software subsystems. (An example is a waveform player component that utilizes the DSP subsystem and data streaming services to effect audio playback functions.)

media component capabilities - The functionality of a media component, including component functions that are supported.

media component type - A class of media components that exhibit similar behavior and capabilities. Examples of media component types are analog video display hardware and MIDI synthesizers.

media control driver (MCD) - A software implementation or method that effects the function of a media component. For OS/2, a media control driver or *media driver* is a dynamic-link library (or set of libraries) that utilizes physical device drivers, Media Device Manager services, and OS/2 to implement the function of the media component.

Media Control Interface (MCI) - A generalized interface to control multimedia devices. Each device has its own MCI driver that implements a standard set of MCI functions. In addition, each media driver can implement functions that are specific to the particular device.

media device - A processor of audiovisual information or media. Media components can be either internal or external physical devices or defined mechanisms for effecting higher-level function from internal hardware and software subsystems. (An example is a waveform player component that utilizes the DSP (Digital Signal Processor) subsystem and data-streaming services to effect audio-playback functions.)

media device capabilities - The functionality of a media component, including supported component functions.

media device connection - A physical or logical link between media component connectors for a particular set of media component instances.

media device connector - A physical or logical input or output on a media component.

media device connector index - An identifier for a media component connector.

media device ID - Media component identification. A unique identifier for a component.

media device instance - A case of an application's use of a media component.

media device manager (MDM) - A system service that, when two or more applications attempt to control a media device, determines which process gains access.

media driver - A device driver for a multimedia device. See also *device driver*.

media driver - A software implementation or method that effects the function of a media device.

media element manager (MEM) - A system service that manipulates multimedia data.

media programming interface (MPI) - A subsystem that provides a comprehensive system programming API layer for multimedia applications.

media segment - An audiovisual object of some type, such as a waveform, song, video clip, and so on.

media unit - A medium on which files are stored; for example, a diskette.

media volume - A (possibly heterogeneous) physical or logical collection of media segments. (Examples are a videodisc, video tape, compact audio disc, OFF file, and RIFF file.)

media volume file - A media volume that is embodied as a conventional binary computer file within a computer file system on storage devices such as disks, diskettes, or CD-ROMs. Such storage devices can be either local or remote. Media volume files can be of various formats, such as OFF or RIFF, and contain segments of various types.

medium shot - A camera angle that reveals more of the subject than a close-up but less than a wide shot, usually from face to waistline; sometimes called a *mid-shot*.

MEM - Media element manager.

MEM IOProc - An internal I/O procedure provided by the MPM/2 system that supports memory files.

memory file - A block of memory that is perceived as a file by an application.

memory playlist - A data structure in the application used to specify the memory addresses to play from or record to. The application can modify the playlist to achieve various effects in controlling the memory stream.

message interface - See *command message interface*.

MIDI Mapper - Provides the ability to translate and redirect MIDI messages to achieve device-independent playback of MIDI sequences.

MIDI message - A sequence of bytes that conform to the MIDI standard. There are two categories: System Exclusive (SysEx) and non-SysEx messages. SysEx messages can be of any length, whereas non-SysEx messages are between one and three bytes.

mid-shot - See *medium shot*.

millisecond - One thousandth of a second.

minutes-seconds-frames (MSF) - A time format based on the 75-frames-per-second CD digital audio standard.

MIPS - Millions of instructions per second. A unit of measure of processing performance equal to one million instructions per second.

mix - The combination of audio or video sources during postproduction.

mixed-media system - Synonym for *multimedia system*.

Mixed Object : Document Content Architecture (MO:DCA) - A data format for multimedia products.

mixer - A device used to simultaneously combine and blend several inputs into one or two outputs.

mixing - (1) In computer graphics, the result of the intersection of two or more colors. (2) In filming, the combining of audio and video sources that is accomplished during postproduction at the *mix*. (3) In recording, the combining of audio sources.

MMIO - Multimedia input/output.

MMIO file services - System services that enable an application to access and manipulate multimedia data files.

MMIO manager - Multimedia input/output manager. The MMIO manager provides services to find, query, and access multimedia data objects. It also supports the functions of memory allocation and file compaction. The MMIO manager uses IOProcs to direct the input and output associated with reading from and writing to different types of storage systems or file formats.

M-Motion - A multimedia platform that offers analog video in addition to quality sound and images. The M-Motion environment consists of the M-Motion Video Adapter/A and the M-Control Program/2 master stream handler.

M-Motion Video Adapter/A - (1) A Micro Channel adapter that receives and processes signals from multiple video and audio sources, and then sends these signals to a monitor and speakers. (2) Dual plane video hardware that offers analog video in addition to quality sound and images. (3) The M-Motion Video Adapter/A requires the M-Control Program/2.

MMPM/2 - Multimedia Presentation Manager/2. See *OS/2 multimedia*.

MMTIME - Standard time and media position format supported by the media control interface. This time unit is 1/3000 second, or 333 microseconds.

MO:DCA - Mixed Object : Document Content Architecture.

mode - A method of operation in which the actions that are available to a user are determined by the state of the system.

model - The conceptual and operational understanding that a person has about something.

module - A language construct that consists of procedures or data declarations and that can interact with other constructs.

moire - An independent, usually shimmering pattern seen when two geometrically regular patterns (as a sampling frequency and a correct frequency) are superimposed. The moire pattern is an alias frequency. See also *aliasing*.

monitor - See *video monitor*.

monitor window - A graphical window, available from a digital video device, which displays the source rectangle, and any subset of this video capture region. See *destination rectangle* for related information.

motion-control photography - A system for using computers to precisely control camera movements so that the different elements of a shot-models and backgrounds, for example-can later be composited with a natural and believable unity.

motion video capture adapter - An adapter that, when attached to a computer, allows an ordinary television picture to be displayed on all or part of the screen, mixing high-resolution computer graphics with video; also enables a video camera to become an input device.

Motion Video Object Content Architecture (MVOCA) - A data format for multimedia products.

Moving Pictures Experts Group (MPEG) - A group that is working to establish a standard for compressing and storing motion video and animation in digital form.

MPEG - Moving Pictures Experts Group.

MPI - Media Programming Interface.

MPI application services - Media Programming Interface application services. Functional services provided by MPI to application programs and higher-level programming constructs, such as multimedia controls.

MS - Script abbreviation for *medium shot*.

MSF - Minutes-seconds-frames.

multimedia - Material presented in a combination of text, graphics, video, image, animation, and sound.

multimedia data object - In an application, an element of a data structure (such as a file, an array, or an operand) that is needed for program execution and that is named or otherwise specified by the allowable character set of the language in which the program is coded.

Multimedia File I/O Services - System services that provide a generalized interface to manipulate multimedia data. The services support buffered and unbuffered file I/O, standard RIFF files, and installable I/O procedures.

multimedia input/output (MMIO) - (1) System services that provide a variety of functions for media file access and manipulation. (2) A consistent programming interface where an application, media driver, or stream handler can refer to multimedia files, read and write data to the files, and query the contents of the files, while remaining independent of the underlying file formats or the storage systems that contain the files.

multimedia navigation system - A tool that gives the information product designer the freedom to link various kinds and pieces of data in a variety of ways so that users can move through it nonsequentially.

multimedia system - (1) A system capable of presenting multimedia material in its entirety. (2) Synonym for *mixed-media system*.

multiple selection - A selection technique in which a user can select any number of objects, or not select any.

Musical Instrument Digital Interface (MIDI) - A protocol that allows a synthesizer to send signals to another synthesizer or to a computer, or a computer to a musical instrument, or a computer to another computer.

mute - To temporarily turn off the audio for the associated medium.

mux - An abbreviation for multiplexer. See also *mixer*.

MVOCA - Motion Video Object Content Architecture.

N

NAPLPS - North American Presentation Level Protocol Syntax.

National Television Standard Committee (NTSC) - A committee that set the standard for color television broadcasting and video in the United States (currently in use also in Japan); also refers to the standard set by this committee (NTSC standard).

node - An abstract term indicating either a node class or a node instance. When used with a class qualifier (for example, application node) it implies an instance (for example, instance of an application class). (RTMIDI-specific term)

node class - A definition of the behavior of a node instance. All instances of the same class are expected to have the same behavior and purpose, although this restriction is not enforced by the driver. (RTMIDI-specific term)

node instance - A vertex in the node network that can receive and transmit MIDI messages. (RTMIDI-specific term)

node network - The collection (graph) of node instances and links. (RTMIDI-specific term)

nondrop time code - Synonym for *full-frame time code*.

noninteractive program - A running program that cannot receive input from the keyboard or other input device.

non-streaming device - (1) A device that contains both source and destination information for multimedia. (2) A device that transmits data (usually analog) directly, without streaming to system memory.

North American Presentation Level Protocol Syntax (NAPLPS) - A protocol used for display and communication of text and graphics in a videotex system; a form of vector graphics.

notebook - A graphical representation that resembles a perfect-bound or spiral-bound notebook that contains pages separated into sections by tabbed divider pages. A user can turn the pages of a notebook to move from one section to another.

NTSC - National Television Standard Committee.

null streaming - (1) The behavior of a stream that can be created and started but which has no associated data flow. (2) The behavior of a stream that can be created and started but which has no associated data flow. For example, a CD-DA is a non-streaming device. (3) A

device that does not stream its data through the MPPM/2 streaming system For example, a CDDA device is a non-streaming device.

O

object - (1) Anything that exists in and occupies space in storage and on which operations can be performed; for example, programs, files, libraries, and folders. (2) Anything to which access is controlled; for example, a file, a program, an area of main storage. (3) See also *data object* and *media object*.

object-action paradigm - A method where users select the object that they want to work with, then choose the action they wish to perform on that object. See *object orientation*.

object class - A categorization or grouping of objects that share similar behaviors and characteristics.

object connection - A link between two objects. Connections can be used for navigation, as with *hypermedia*, or for data transfer between objects.

Object Content Architecture (OCA) - A data format for multimedia products.

object decomposition - The process of breaking an object into its component parts.

object orientation - An orientation in a user interface in which a user's attention is directed toward the objects the user works with, rather than applications, to perform a task.

object-oriented user interface - A type of user interface that implements object orientation and the object-action paradigm.

object template - An object that can be used to create another object of the same object class. The template is a basic framework of the object class, and the newly created object is an instance of the object class.

OCA - Object Content Architecture.

OEM - Original equipment manufacturer.

OFF - Operational file format.

offline edit - A preliminary or test edit usually done on a low-cost editing system using videocassette work tapes. (An offline edit is done so that decisions can be made and approvals given prior to the final edit.)

online edit - The final edit, using the master tapes to produce a finished program.

operational file format (OFF) - A file format standard.

optical disc - A disc with a plastic coating on which information (as sound or visual images) is recorded digitally as tiny pits and read using a laser. The three categories of optical discs are CD-ROM, WORM, and erasable.

optical drive - Drives that run optical discs.

optical reflective disc - A designation of the means by which the laser beam reads data on an optical videodisc. In the case of a reflective disc, the laser beam is reflected off a shiny surface on the disc.

opticals - Visual effects produced optically by means of a device (an optical printer) that contains one camera head and several projectors. The projectors are precisely aligned so as to produce multiple exposures in exact registration on the film as in the camera head.

original footage - The footage from which the program is constructed.

OS/2 multimedia - A subsystem service of OS/2 that provides a software platform for multimedia applications. It defines standard interfaces between multimedia devices and OS/2 multimedia applications.

overlay - The ability to superimpose text and graphics over video.

overlay device - Provides support for video overlaying along with video attribute elements. The video overlaying handles tasks such as displaying, and sizing video. Synonym for *video overlay device*.

P

PAL - Phase Alternation Line.

palette - See *color palette*, *standard palette*, and *custom palette*.

pan - A camera movement where the camera moves sideways on its stationary tripod; left-to-right balance in an audio system.

panel - A particular arrangement of information grouped together for presentation to users in a window.

panning - Progressively translating an entire display image to give the visual impression of lateral movement of the image.

In computer graphics, the viewing of an image that is too large to fit on a single screen by moving from one part of the image to another.

paradigm - An example, pattern, or model.

patch mapping - The reassignment of an instrument patch number associated with a specific synthesizer to the corresponding standard patch number in the General MIDI specification.

pause - To temporarily halt the medium. The halted visual should remain displayed but no audio should be played.

pause stop - In data streaming, a stop that pauses the data stream but does not disturb any data.

PDC - Physical device component.

PDD - Physical device driver.

pedestal up/down - A camera movement where the camera glides up or down on a boom.

pel - The dimensions of a toned area at a picture element. See also *picture element*.

Phase Alternation Line (PAL) - Television broadcast standard for European video outside of France and the Soviet Union.

physical device driver (PDD) - A program that handles hardware interrupts and supports a set of input and output functions.

picon - A graphic or natural image reduced to icon size. Similar to *thumbnail*.

picture element - In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. See also *pel*.

picture-in-picture - A video window within a larger video window.

pixel - See *picture element*.

plaintext - Nonencrypted data.

platform - In computer technology, the principles on which an operating system is based.

play backward - To play the medium in the backward direction.

playback window - The graphic window in which software motion video is displayed. This window can be supplied by an application, or a default window can be created by a digital video device.

play forward - To play the medium in the forward direction.

pointer - A symbol, usually in the shape of an arrow, that a user can move with a pointing device. Users place the pointer over objects they want to work with.

pointing device - A device, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

polish - The version of the script submitted for final approval.

polyphony - A synthesizer mode where more than 1 note can be played at a time. Most synthesizers are 16-note to 32-note polyphonic.

postproduction - The online and offline editing process.

PPQN - (1) Parts-per-quarter-note. (2) A time format used in musical instrument digital interface (MIDI).

preproduction - The preparation stage for video production, when all logistics are planned and prepared.

preroll - The process of preparing a device to begin a playback or recording function with minimal latency. During a multimedia sequence, it might require that two devices be cued (prerolled) to start playing and recording at the same time.

primary window - A window in which the main interaction between a user and an object takes place.

Proc - A custom procedure, called by the particular utility manager, to handle input or output to files of a format different from DOS, MEM, or BND; for example, AVC or TIFF. By installing custom procedures, existing applications no longer need to store multiple copies of the same media file for running on various platforms using different file formats. See also *static resource* and *dynamic resource*.

production - In videotaping, the actual shooting.

production control room - The room or location where the monitoring and switching equipment is placed for the direction and control of a television production.

progress indicator - A control, usually a read-only slider, that informs the user about the status of a user request.

props - In videotaping, support material for the shoot, for example, equipment being promoted, auxiliary equipment, software, or supplies; anything provided to make the set look realistic and attractive.

protection master - A copy of the edit master that is stored as a backup.

PS/2 CD-ROM-II Drive - An IBM CD-ROM drive that can play compact disc digital audio (CD-DA) and CD-ROM/XA interleaved audio, video, and text, and adheres to the Small Computer System Interface (SCSI). The drive can be installed on Micro Channel and non-Micro Channel IBM PS/2 systems.

pulse code modulation (PCM) - In data communication, variation of a digital signal to represent information.

push button - A graphical control, labeled with text, graphics, or both, that represents an action that will be initiated when a user selects it. For example, when a user clicks on a *Play* button, a media object begins playing.

R

raster graphics - Computer graphics in which a display image is composed of an array of pels arranged in rows and columns.

raw footage - Synonym for *original footage*.

ray-tracing - A technique used by 3-D rendering software programs that automatically figures an object's position in three dimensions and calculates shadows, reflections, and hidden surfaces based on user-entered light locations and material characteristics. (In other words, if the user orders an object to be a mirror, the computer produces the mirror with all its correct reflective properties.)

real time - (1) Pertaining to the processing of data by a computer in connection with another process outside the computer according to time requirements imposed by the outside process. This term is also used to describe systems operating in conversational mode and processes that can be influenced by human intervention while they are in progress. (2) A process control system or a computer-assisted instruction program, in which response to input is fast enough to affect subsequent output.

real-time recording - Refers to the capturing of video and audio data in real time, as the analog signals are generated from the video source device. The video source device can be a camcorder, or a videotape or videodisc player.

record - To transfer data from one source (for example, microphone, CD, videodisc) or set of sources to another medium.

Redbook audio - The storage format of standard audio CDs. See also *compact disc*, *digital audio (CD-DA)*.

reference frame - (1) Refers to the complete frame that is created at periodic intervals in the output stream. An editing operation always begins at a reference frame. (2) Synonymous with *key frame* and *I-frame*. (3) See *delta frame*.

reflective disc - See *optical reflective disc*.

render - In videotaping, to create a realistic image from objects and light data in a scene.

repeat - A mode which causes the medium to go to the beginning and start replaying when it reaches the medium's end.

resolution - (1) In computer graphics, a measure of the sharpness of an image, expressed as the number of lines and columns on the display screen or the number of pels per unit of area. (2) The number of lines in an image that an imaging system (for example, a

telescope, the human eye, a camera, and so on) can resolve. A higher resolution makes text and graphics appear clearer.

resource - As used in the multimedia operating system, any specific unit of data created or used by a multimedia program. See also *static resource* and *dynamic resource*.

resource handler - A system service that loads, saves, and manipulates multimedia program units of data.

resource interchange file format (RIFF) - A tagged file format framework intended to be the basis for defining new file formats.

resync - Recovery processing performed by sync-point services when the failure of a session, transaction program, or LU occurs during sync-point processing. The purpose of resync is to return protected resources to consistent states.

resync tolerance value - A minimum time difference expressed in MMTIME format.

Revisable Form Text : Document Content Architecture (RFT:DCA) - A data format for multimedia products.

rewind - To advance the medium in the backward direction quickly, and optionally allow the user to scan the medium.

RFT:DCA - Revisable Form Text : Document Content Architecture.

RGB - Color coding where the brightness of the additive primary colors of light, red, green, and blue, are specified as three distinct values of white light.

RIFF - Resource interchange file format.

RIFF chunk - A chunk with a chunk ID of *RIFF*. In a RIFF file, this must be the first chunk.

RIFF compound file - A file containing multiple file elements or one file element that makes up the entire RIFF file. The MMIO manager provides services to find, query, and access any file elements in a RIFF compound file. Synonym for *bundle file*.

rotoscope - A camera setup that projects live-action film, one frame at a time, onto a surface so that an animator can trace complicated movements. When filmed, the completed animation matches the motion of the original action.

rough cut - (1) The result of the offline edit. (2) A video program that includes the appropriate footage in the correct order but does not include special effects.

RTMIDI - Real-time MIDI subsystem.

S

SAA - Systems Application Architecture.

safety - An extra shot of a scene that is taken as a backup after an acceptable shot (the *buy*) has been acquired.

saturation - The amounts of color and grayness in a hue that affect its vividness; that is, a hue with high saturation contains more color and less gray than a hue with low saturation. See also *hue*.

sampler - A device that converts real sound into digital information for storage on a computer.

scan backward - To display the video and optionally play the audio while the medium is advancing in the backward direction rapidly.

scan converter - A device that converts digital signal to NTSC or PAL format.

scan forward - To display the video and optionally play the audio while the medium is advancing in the forward direction rapidly.

scan line - (1) In a laser printer, one horizontal sweep of the laser beam across the photoconductor. (2) A single row of picture elements.

scanner - A device that examines a spatial pattern, one part after another, and generates analog or digital signals corresponding to the pattern.

SCB - Subsystem control block.

scene - A portion of video captured by the camera in one continuous shot. The scene is shot repeatedly (each attempt is called a *take*) until an acceptable version, called the *buy*, is taken.

scheduler - The code responsible for passing messages to instances. The scheduler has two queues, one for normal real-time MIDI messages (Q1), and the other for low-priority SysEx messages (Q2). (RTMIDI-specific term)

scheduler daemon - A small executable program, MIDIDMON.EXE, which is used to provide deferred-interrupt processing for the scheduler. The daemon is a super high-priority thread which gets blocked in ring 0. When the scheduler needs to run, this thread is unblocked. When the scheduler is finished, it re-blocks itself. When the unblocking occurs during an interrupt, the OS/2 kernel runs the daemon thread immediately after the interrupt handler has exited. This approach guarantees that the scheduler runs at task time.

scripting - Writing needed dialog.

scroll bar - A window component that shows a user that more information is available in a particular direction and can be scrolled into view. Scroll bars should not be used to represent an analog setting, like volume. Sliders should be used.

SCSI - Small computer system interface.

SECAM - Sequential Couleurs a Memoire. The French standard for color television.

secondary window - A window that contains information that is dependent on information in a primary window and is used to supplement the interaction in the primary window.

secondary window manager - A sizable dialog manager that enables application writers to use CUA-defined secondary windows instead of dialog boxes.

second generation - A direct copy from the master or original tape.

selection - The act of explicitly identifying one or more objects to which a subsequent choice will apply.

selection technique - The method by which users indicate objects on the interface that they want to work with.

semaphore - (1) A variable that is used to enforce mutual exclusion. (T) (2) An indicator used to control access to a file; for example, in a multiuser application, a flag that prevents simultaneous access to a file.

sequencer - A digital tape recorder.

set - In videotaping, the basic background or area for production.

settings - Characteristics of objects that can be viewed and sometimes altered by the user. Examples of a file's settings include name, size, and creation date. Examples of video clip's settings include brightness, contrast, color, and tint.

settings view - A view of an object that provides a way to change characteristics and options associated with the object.

SFX - Script abbreviation for *special effects*.

shade - To darken with, or as if with, a shadow; to add shading to.

sharpness - Refers to the clarity and detail of a video image. A sharpness value of 0 causes the video to be generally fuzzy with little detail. A sharpness value of 100 causes the video to be generally very detailed, and may appear grainy.

SHC - Stream handler command.

shoot - To videotape the needed pictures for the production.

shooting script - Synonym for *final script*.

shot list - A list containing each shot needed to complete a production, usually broken down into a schedule.

simple device - A multimedia device model for hardware which does not require any additional objects, known as device elements, to perform multimedia functions.

sine wave - A waveform that represents periodic oscillations of a pure frequency.

single plane video system - Refers to when video and graphics are combined into one buffer. This may appear the same as a dual plane video system, but since all the data is in one buffer, capture and restore operations will obtain both graphics and video components in one operation. See also *dual plane video system*.

single selection - A selection technique in which a user selects one, and only one, item at a time.

slave stream - A stream that is dependent on the *master stream* to maintain synchronization.

slave stream handler - In SPI, regularly updates the sync pulse EVCB with the stream time. The Sync/Stream Manager checks the slave stream handler time against the master stream time to determine whether to send a sync pulse to the slave stream handler.

slider - A visual component of a user interface that represents a quantity and its relationship to the range of possible values for that quantity.

A user can also change the value of the quantity. Sliders are used for volume and time control.

slider arm - The visual indicator in the slider that a user can move to change the numerical value.

slider button - A button on a slider that a user clicks on to move the slider arm one increment in a particular direction, as indicated by the directional arrow on the button.

slide-show presentation - Synonym for *storyboard*.

slot - A distinct position in an instance from which links can be attached. The same message is sent along all links on a slot, but an instance can determine at run-time on which slots the message should be sent. An instance can support multiple slots if it wants to be able to send different messages to different targets. (RTMIDI-specific term)

small computer system interface (SCSI) - An input and output bus that provides a standard interface between the OS/2* multimedia system and peripheral devices.

SMH - Stream manager helper.

SMPTE - Society of Motion Picture and Television Engineers.

SMPTE time code - A frame-numbering system developed by SMPTE that assigns a number to each frame of video. The 8-digit code is in the form HH:MM:SS:FF (hours, minutes, seconds, frame number). The numbers track elapsed hours, minutes, seconds, and frames from any chosen point.

SMV - Software motion video.

socketable user interface - An interface defined by multimedia controls that enable the interface to be plugged into and unplugged from applications without affecting the underlying object control subsystem.

sound track - Synonym for *audio track*.

source node - An instance which can generate a compound message. Hardware nodes generate messages from data received from Type A drivers. Application nodes generate them from data sent from an application. (RTMIDI-specific term)

source rectangle - An abstract region representing the area available for use by a video capture adapter. This window is displayed in the monitor window of the digital video device. A subset of the maximum possible region to be captured can be defined; such a subset is shown by an animated dashed rectangle in the monitor window.

source window - See *source rectangle*.

SPCB - Stream protocol control block.

special effects - In videotaping, any activity that is not live footage, such as digital effects, computer manipulation of the picture, and nonbackground music.

SPI - Stream programming interface.

split streaming - A mechanism provided by the Sync/Stream Manager to create one data stream source with multiple targets.

SPP - A time format based on the number of beats-per-minute in the MIDI file.

sprite - An animated object that moves around the screen without affecting the background.

sprite graphics - A small graphics picture, or series of pictures, that can be moved independently around the screen, producing animated effects.

squeeze-zoom - A DVE where one picture is reduced in size and displayed with a full-screen picture.

SSM - Sync/Stream Manager.

standard multimedia device controls - These controls provide the application developer with a CUA compliant interface for controlling audio attributes, video attributes, and videodisc players. These controls simplify the programming task required to create the interface and handle the presentation of the interface and all interaction with the user. They also send the Media Control Interface (MCI) commands to the Media Device Manager (MDM) for processing.

standard objects - A set of common, cross-product objects provided and supported by the system. Examples include folders, printers, shredders, and media players.

standard palette - A set of colors that is common between applications or images. See also *custom palette* and *color palette*.

static resource - A *resource* that resides on any read-and-write or read-only medium. Contrast with *dynamic resource*.

status area - Provides information as to the state of the medium and device, or both. It should indicate what button is currently pressed and

what modes (for example, mute) are active.

step backward - To move the medium backward one frame or segment at a time.

step forward - To move the medium forward one frame or segment at a time.

still - A static photograph.

still image - See *video image*.

still video capture adapter - An adapter that, when attached to a computer, enables a video camera to become an input device. See also *motion video capture adapter*.

stop - Halt (stops) the medium.

storage system - The method or format a functional unit uses to retain or retrieve data placed within the unit.

storage system IOProc - A procedure that unwraps data objects such as RIFF files, RIFF compound files, and AVC files. IOProcs are ignorant of the content of the data they contain. A storage system IOProc goes directly to the OS/2 file system (or to memory in the case of a MEM file) and does not pass information to any other file format or storage system IOProc. The internal I/O procedures provided for DOS files, memory files, and RIFF compound files are examples of storage system IOProcs, because they operate on the storage mechanism rather than on the data itself. See also *file format IOProc*.

storyboard - (1) A visual representation of the script, showing a picture of each scene and describing its corresponding audio. (2) Synonym for *slide-show presentation*.

storyboarding - Producing a sequence of still images, such as titles, graphics, and images, to work out the visual details of a script.

stream - To send data from source to destination via buffered system memory.

stream connector - A port or connector that a device uses to send or receive. See also *connector*.

stream handler - A routine that controls a program's reaction to a specific external event through a continuous string of individual data values.

stream handler command (SHC) - Synchronous calls provided by both ring 3 DLL stream handlers as a DLL call and by ring 0 PDD stream handlers as a IDC call. The stream handler commands are provided through a single entry point, SHCEntryPoint, which accepts a parameter structure on input. This enables the DLL and PDD interfaces to the stream manager to be the same.

stream manager - A system service that controls the registration and activities of all stream handlers.

stream manager helper (SMH) - Routines provided by the stream manager for use by all stream handlers. The stream handlers use these helper routines to register with the manager, report events, and synchronize cues to the manager to request or return buffers to the manager. They are synchronous functions and are available to both ring 3 DLL stream handlers as a DLL call and to ring 0 PDD stream handlers.

stream programming interface - A system service that supports continual flow of data between physical devices.

stream programming interface (SPI) - A system service that supports continual flow of data between physical devices.

stream protocol control block (SPCB) - The system service that controls the behavior of a specified stream type. This enables you to subclass a stream's data type, change data buffering characteristics, and alter synchronization behavior and other stream events.

strike - In videotaping, to clear away, remove, or dismantle anything on the set.

subchunk - The first *chunk* in a RIFF file is a *RIFF chunk*; all other chunks in the file are subchunks of the RIFF chunk.

subclassing - The act of intercepting messages and passing them on to their original intended recipient.

super - Titles or graphics overlaid on the picture electronically. See also *superimpose*.

superimpose - To overlay titles or graphics on the picture electronically.

S-video - (1) Separated video or super video. (2) A signal system using a Y/C format. (3) See also *Y/C*, *composite video*, and *component video*.

S-Video input connector - A special connector that separates the chrominance from the luminance signal.

sweetening - (1) The equalization of audio to eliminate noise and obtain the cleanest and most level sound possible. (2) The addition of laughter to an audio track.

switching - Electronically designating, from between two or more video sources, which source's pictures are recorded on tape. Switching can occur during a shoot or during an edit.

symmetric video compression - A technology in which the computer can be used to create, as well as play back, full-motion, full-color video.

sync - Synchronization or synchronized.

synchronization - The action of forcing certain points in the execution sequences of two or more asynchronous procedures to coincide in time.

synchronous - Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals.

sync group - A *master stream* and all its *slaves* that can be started, stopped, and searched as a group by using the slaves flag on each of the following SPI functions:

- SpiStartStream
- SpiStopStream
- SpiSeekStream

sync pulse - A system service that enables each slave stream handler to adjust the activity of that stream so that synchronization can be maintained. Sync pulses are introduced by transmission equipment into the receiving equipment to keep the two equipments operating in step.

sync signal - Video signal used to synchronize video equipment.

synthesizer - A musical instrument that allows its user to produce and control electronically generated sounds.

system message - A predefined message sent by the MMIO manager for the message's associated function. For example, when an application calls mmioOpen, the MMIO manager sends an MMIOM_OPEN message to an I/O procedure to open the specified file.

Systems Application Architecture (SAA) - A set of IBM software interfaces, conventions, and protocols that provide a framework for designing and developing applications that are consistent across systems.

T

tagged image file format (TIFF) - An easily transportable image file type used by a wide range of multimedia software.

take - During the shoot in videotaping, each separate attempt at shooting a scene. This is expressed as: Scene 1, Take 1; Scene 1, Take 2, and so on.

talent - On-screen person (professional or amateur) who appears before the camera or does voice-over narration.

TAM - Telephone answering machine.

target node - An instance which receives a message but does not forward it because it is the final instance in a chain of processing. For example, a hardware node is a target node because when it receives a message, it sends it to another device driver, and not to another instance. (RTMIDI-specific term)

tearing - Refers to when video is displaced horizontally. This may be caused by sync problems.

TelePrompter - A special monitor mounted in front of a camera so that talent can read text and will appear to be looking at the camera.

thaw - See *unfreeze*.

thumbnail - A small representation of an object. For example, a full screen image might be presented in a much smaller area in an authoring system time line. A *picon* is an example of a thumbnail.

TIFF - Tagged Image File Format time code.

tilt - A camera movement where the camera pivots up or down on its stationary tripod.

timbre - The distinctive tone of a musical instrument or human voice that distinguishes it from other sounds.

time code - See *SMPTE time code*.

time-line processor - A type of authoring facility that displays an event as elements that represent time from the start of the event.

tint - See *hue*.

TMSF - A time format expressed in tracks, minutes, seconds, and frames, which is used primarily by compact disc audio devices.

tone (bass, treble, etc...) - A control that adjusts the various attributes of the audio.

tool palette - A palette containing choices that represent tools, often used in media editors (such as graphics and audio editors). For example, a user might select a "pencil" choice from the tool palette to draw a line in the window.

touch area - (1) An area of a display screen that is activated to accept user input. (2) Synonymous with *anchor, hot spot, and trigger*.

track - A path associated with a single Read/Write head as the data medium moves past it.

track advance - To advance the medium to the beginning of the next track.

track reverse - To rewind the medium to the beginning of the current track. If it is at the beginning of the track it will then jump to the beginning of the previous track.

transform device - A device that modifies a signal or stream received from a transport device. Examples are amplifier-mixer and overlay devices.

translator - A computer program that can translate. In telephone equipment the device that converts dialed digits into call-routine information.

transparency - Refers to when a selected color on a graphics screen is made transparent to allow the video "behind it" to become visible. Often found in dual plane video subsystems.

transparent color - Video information is considered as being present on the video plane which is maintained behind the graphics plane. When an area on the graphics plane is painted with a transparent color, the video information in the video plane is made visible. See also *dual plane video system*.

transport device - A device that plays, records, and positions a media element, and either presents the result directly or sends the material to a *transform device*. Examples are videodisc players, CD-ROMs, and digital audio (wave) player.

treatment - A detailed design document of the video.

tremolo - A vibrating effect of a musical instrument produced by small and rapid amplitude variations to produce special musical effects.

trigger - (1) An area of a display screen that is activated to accept user input. (2) Synonymous with *anchor, hot spot, and touch area*.

truck - In videotaping, a sideways camera movement of the tripod on which the camera is mounted.

tweening - (1) The process of having the computer draw intermediate animation frames between key frames. In other words, the animation tool requires only that pictures of key sections of a motion be provided; the software calculates all the in-between movements. (2) Synonym for *in-betweening*.

U

Ultimatte - The trade name of a very high-quality, special-effects system used for background replacement and image composites.

U-matic - A video cassette system using 0.75-inch tape format.

underrun - Loss of data caused by the inability of a transmitting device or channel to provide data to the communication control logic (SDLC or BSC/SS) at a rate fast enough for the attached data link or loop.

unfreeze - (1) To return to action after a *freeze*. (2) Enables updates to the video buffer. (3) Synonym for *thaw*.

unidirectional microphone - A microphone that responds to sound from only one direction and is not subject to change of direction. (A unidirectional microphone is the type of microphone employed in computers capable of voice recognition.)

unload - To eject the medium from the device.

user-defined message - A private message sent directly to an I/O procedure by using the `mmioSendMessage` function. All messages begin with an MMIOM prefix, with user-defined messages starting at `MMIOM_USER` or above.

user interface - The area at which a user and an object come together to interact. As applied to computers, the ensemble of hardware and software that allows a user to interact with a computer.

user's conceptual model - A user's mental model about how things should work. Much of the concepts and expectations that make up the model are derived from the user's experience with real-world objects of similar type, and experience with other computer systems.

V

value set - A control used to present a series of mutually exclusive graphical choices. A tool palette in a paint program can be implemented using a value set.

VCR - Videocassette recorder.

VDD - Virtual device driver.

VDH - Virtual device helper.

VDP - Video display processor.

vector graphics - See *coordinate graphics*.

vendor specific drivers - An extension to an MCD to execute hardware specific commands.

VHS - Very high speed. A consumer and industrial tape format (VHS format).

vicon - A vicon, or video icon, can be an animation or motion video segment in icon size. Usually this would be a short, repeating segment, such as an animation of a cassette tape with turning wheels.

video - Pertaining to the portion of recorded information that can be seen.

video aspect ratio - See *aspect ratio*.

video attribute control - Provides access to and operation of the standard video attributes: brightness, contrast, freeze, hue, saturation, and sharpness. All device communication and user interface support is handled by the control.

video attributes - Refers to the standard video attributes: brightness, contrast, freeze, hue, saturation, and sharpness.

video clip - A section of filmed or videotaped material.

video clipping - See *clipping*.

video digitizer - Any system for converting analog video material to digital representation. (For example, see *DVI*.)

video display buffer - The buffer containing the visual information to be displayed. This buffer is read by the video display controller.

video display controller - The graphics or video adapter that connects to a display and presents visual information.

video encoder - A device (adapter) that transforms the high-resolution digital image from the computer into a standard television signal, thereby allowing the computer to create graphics for use in video production.

video graphics adapter - A graphics controller for color displays. The pel resolution of the video graphics adapter is 4:4.

video image - (1) A still video image that has been captured. (2) Synonymous with *image* and *still image*.

video monitor - A display device capable of accepting a video signal that is not modulated for broadcast either on cable or over the air; in videotaping, a television screen located away from the set where the footage can be viewed as it is being recorded.

video overlay - See *overlay*.

video overlay device - See *overlay device*.

video plane - In a dual plane video system, the video plane contains the video. This video plane will be combined with the graphics plane to create an entire display image.

video programming interface (VPI) - A subsystem that performs output from video source to video window.

video quality - The compression quality level setting to be set for the CODEC. This value is in the range of 0 (min) - 100 (max).

video record rate - Frame rate for recording as an integral number of frames per second. This sets the target capture rate, but there are no assurances this rate will be attained. Drop frame records will be inserted into the output data stream to indicate frames dropped during the capture/record process.

video record frame duration - Frame rate for recording as the time duration of each frame in microseconds. Useful for setting non-integer frame rates, for example, 12.5 FPS of a PAL videodisc: $1000000/12.5 = 8000$ microseconds.

video signal - An electrical signal containing video information. The signal must be in some standard format, such as NTSC or PAL.

VSD - Vendor Specific Driver

video scaling - (1) Expanding or reducing video information in size or area. (2) See also *aspect ratio*.

video scan converter - A device that emits a video signal in one standard into another device of different resolution or scan rate.

video segment - A contiguous set of recorded data from a video track. A video segment might or might not be associated with an audio segment.

video signal - An electrical signal containing video information. The signal must be in some standard format, such as NTSC or PAL.

video source selection - The ability of an application to change to an alternate video input using the **connector** command.

video tearing - See *tearing*.

video teleconferencing - A means of telecommunication characterized by audio and video transmission, usually involving several parties. Desktop video teleconferencing could involve having the audio and video processed by the user's computer system, that is, with the other users' voices coming through the computer's speaker, and video windows of the other users displayed on the computer's screen.

videocassette recorder (VCR) - A device for recording or playing back videocassettes.

videodisc - A disc on which programs have been recorded for playback on a computer (or a television set); a recording on a videodisc. The most common format in the United States and Japan is an NTSC signal recorded on the optical reflective format.

videodisc player control - Provides access to and operation of the following videodisc functions: eject, pause, play forward, play reverse, position, record, repeat, rewind, scan forward, scan reverse, step forward, step reverse, and stop. All device communication and user interface support is handled by the control.

videotape - (1) The tape used to record visual images and sound. (2) To make a videotape of. (3) A recording of visual images and sound made on magnetic tape. (All shooting is done in this format, even if the results are later transferred to videodisc or film.)

videotape recorder (VTR) - A device for recording and playing back videotapes. (The professional counterpart of a consumer VCR.)

videotex - A system that provides two-way interactive information services, including the exchange of alphanumeric and graphic information, over common carrier facilities to a mass consumer market using modified TV displays with special decoders and modems.

video windows - Graphical PM-style windows in which video is displayed. Most often associated with the video overlay device.

view - The form in which an object is presented. The four kinds of views are: composed, contents, settings, and help.

viewport - An area on the usable area of the display surface over which the developer has control of the size, location, and scaling, and in which the user can view all or a portion of the data outlined by the window.

virtual device helper - A system service that is available to perform essential functions.

VO - Script abbreviation for *voice-over*.

voice-over - (1) The voice of an unseen narrator in a video presentation. (2) A voice indicating the thoughts of a visible character without the character's lips moving.

volume - The intensity of sound. A volume of 0 is minimum volume. A volume of 100 is maximum volume.

VPI - Video programming interface.

VTR - Videotape recorder.

W

walk-through - A type of animated presentation that simulates a walking tour of a three-dimensional scene.

walk-up-and-use interface - An interface that the target audience should be able to use without having to read manuals or instructions, even if they have never seen the interface.

waveform - (1) A graphic representation of the shape of a wave that indicates its characteristics (such as frequency and amplitude). (2) A digital method of storing and manipulating audio data.

wide shot - Synonym for *long shot*.

wild footage - Synonym for *original footage*.

window - An area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen.

window coordinates - The size and location of a window.

wipe - Technical effect of fading away one screen to reveal another.

workplace - A container that fills the entire screen and holds all of the objects that make up the user interface.

write once/read many (WORM) - Describes an optical disc that once written to, cannot be overwritten. Storage capacity ranges from 400MB to 3.2GB. Present technology allows only one side to be read at a time; to access the other side, the disk must be turned over.

WS - Script abbreviation for *wide shot*.

WYSIWYG - What You See Is What You Get. The appearance of the object is in actual form. For example, a document that looks the same on a display screen as it does when it is printed. Composed views of objects are often WYSIWYG.

X Y Z

Y - Refers to the luminance portion of a Y/C video signal.

Y/C - Color image encoding that separates luminance ((Y) and *chrominance* (C) signals.

YIQ - Image encoding scheme similar to YUV that selects the direction of the two color axes, I and Q, to align with natural images. As an average, the I signal bears much more information than the Q signal. (YIQ is used in the NTSC video standard.)

YUV - Color image encoding scheme that separates luminance (Y) and two color signals: red minus Y (U), and blue minus Y (V). Transmission of YUV can take advantage of the eye's greater sensitivity to luminance detail than color detail.

zoom in - An optical camera change where the camera appears to approach the subject it is shooting.

zooming - The progressive scaling of an image in order to give the visual impression of movement of all or part of a display group toward or away from an observer.

zoom out - An optical camera change where the camera appears to back up from the subject it is shooting.
